INTERLEAVING ALLOCATION, PLANNING, AND SCHEDULING FOR HETEROGENEOUS MULTI-ROBOT COORDINATION THROUGH SHARED CONSTRAINTS

A Dissertation
Presented to
The Academic Faculty

By

Andrew Messing

In Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy in
Robotics
from the
School of Interactive Computing

Georgia Institute of Technology

December 2022

INTERLEAVING ALLOCATION, PLANNING, AND SCHEDULING FOR HETEROGENEOUS MULTI-ROBOT COORDINATION THROUGH SHARED CONSTRAINTS

Thesis committee:

Dr. Seth Hutchinson, Advisor
School of Interactive Computing
Georgia Institute of Technology

Dr. Nicholas Roy
Department of Aeronautics and Astronautics
Massachusetts Institute of Technology

Dr. Sonia Chernova
School of Interactive Computing
Georgia Institute of Technology

Dr. Aleksandra Faust
Senior Staff Research Scientist
Google Brain Research

Dr. Harish Ravichandar
School of Interactive Computing
Georgia Institute of Technology

Date approved: Nov. 29, 2022

ACKNOWLEDGMENTS

I would like to begin by thanking my advisor – Dr. Seth Hutchinson. His guidance on everything from connecting with collaborators to presentation of research was instrumental in the success of my Ph.D. journey. He continually challenged me to consider more of the reasons behind the success of various algorithms and frameworks, and I think the result is much improved and more accessible because of it. His vast knowledge of the field of robotics and its current state have been extremely helpful during my exploration process that led to this dissertation.

Next, I would like to thank the rest of my committee – Dr. Sonia Chernova, Dr. Harish Ravichadar, Dr. Nicholas Roy, and Dr. Aleksandra Faust. Through many discussions and helpful feedback, I learned much from each of you that has aided me through this journey and in developing the research, and its presentation, that is contained in this dissertation.

Additionally, I would like to thank my collaborators. Glen Neville and I had numerous (almost daily) conversations over the years coming up with, developing, and nitpicking the details of what would turn into the framework that is the central part of this dissertation in GRSTAPS and its extensions. Jacopo Banfi, Martina Stadler, and I had many discussions, where they provided new perspectives, approaches, and algorithms. From all of this, I have learned a great deal and our works were much better because of it.

Also, I would like to thank my cohort. Before I started my Ph.D. journey, a coworker recommended that I be careful about my work/life balance due the stereotype surrounding Ph.D. students. Thanks to this group, this was never a concern. Between going to sporting events, traveling, hosting social events, and playing intramurals, I think we balanced progressing the state-of-the-art and enjoying life extremely well – even through the unknowns of Covid. Furthermore, the always available support and advice is very appreciated. Going through the process in parallel with friends, from start to finish, makes everything more enjoyable and helps to keep one's sanity.

Finally, I would like to thank my family. I first told them that I planned to get a Ph.D. in robotics twelve years ago and they have constantly supported and encouraged me every step of the way. Furthermore, they taught me the importance of hard-work, persistence, patience, and so many more skills that have allowed me to be successful and get to where I am today.

TABLE OF CONTENTS

Acknowledgments iii

List of Tables xi

List of Figures xii

List of Acronyms xv

Summary xviii

Chapter 1: Introduction 1

1.1 Thesis Statement 3

1.2 Contributions 4

1.3 Relevant Publications 6

1.4 Dissertation Organization 7

Chapter 2: Interleaving Task Planning and Scheduling 9

2.1 Introduction 9

2.2 Background 11

2.2.1 Task Planning 12

2.2.2 Scheduling 16

2.2.3 Temporal Planning 16

2.2.4 Hierarchical Planning 16
2.2.5 Multi-Agent Planning 17
2.3 Related Work 18
2.4 Forward Chaining Partial-Order Planner 19
2.5 Forward Chaining Hierarchical Partial-Order Planner 23
2.6 Experimental Evaluations 26
2.6.1 Forward Chaining Partial-Order Planner 26
2.6.2 Forward Chaining Hierarchical Partial-Order Planner 29
2.7 Discussion & Conclusion 32

Chapter 3: Interleaving Task Allocation, Scheduling, and Motion Planning . . . 34
3.1 Introduction 34
3.2 Related Work 36
3.3 Problem Description 37
3.3.1 Problem Domain 38
3.3.2 Solution Specification 39
3.4 Communication Between Sub-Problems 40
3.5 Incremental Task Allocation Graph Search 41
3.5.1 Task Allocation 42
3.5.2 Scheduling and Motion Planning 46
3.6 Experimental Evaluations 49
3.6.1 Relative Influence of APR and NSQ on Performance 50
3.6.2 Effects of Interleaving on Performance 51

3.6.3 Comparison against CFLA2 and CCF 52

3.6.4 Summary 55

3.7 Discussion & Conclusion 56

Chapter 4: Interleaving Allocation, Planning, and Scheduling 58

4.1 Introduction 58

4.2 Related Work 60

4.2.1 Multi-Agent Planning 60

4.2.2 Simultaneous Task Allocation and Planning 62

4.2.3 Task and Motion Planning 64

4.2.4 Multi-Vehicle Routing 66

4.3 Problem Formulation 68

4.3.1 Temporal Planning 68

4.3.2 Trait-based Time-extended Task Allocation 69

4.3.3 Scheduling 71

4.3.4 Motion Planning 72

4.3.5 Simultaneous Task Allocation and Planning with Spatiotemporal Constraints 73

4.4 Communication Between Sub-Problems 74

4.5 Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling 75

4.5.1 Algorithmic Assumptions 76

4.5.2 Task Planning 77

4.5.3 Task Allocation 81

4.5.4 Scheduling 86
4.5.5 Motion Planning 90
4.6 Experimental Evaluations 91
4.6.1 Metrics 93
4.6.2 Comparisons with Sequential Baselines 93
4.6.3 Comparisons with Temporal Planner Baselines 100
4.6.4 Testing the limits 105
4.7 Discussion & Conclusion 106

Chapter 5: Heterogeneous Coalition Scheduling with Temporal Uncertainty . . 109

5.1 Introduction 109
5.2 Related Work 111
5.2.1 Coalition Scheduling 111
5.2.2 Scheduling with Temporal Uncertainty 112
5.3 Problem Description 114
5.3.1 Heterogeneous Coalition Scheduling Problem 114
5.3.2 Heterogeneous Coalition Scheduling with Temporal Uncertainty Problem 117
5.4 Approach 118
5.4.1 Sample Average Approximation 118
5.4.2 Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee 120
5.4.3 SPRT for α-robustness guarantee 123
5.5 Experimental Evaluations 125

5.5.1 Impact of the risk tolerance . . . 127
5.5.2 Comparison with other Scheduling Approaches . . . 128
5.5.3 Usage within the GRSTAPS Framework . . . 131
5.5.4 Limitations . . . 133
5.6 Discussion & Conclusion . . . 134
5.7 Appendix: Sequential Probability Ratio Test . . . 135

Chapter 6: Learning to Set Task Orderings for Heterogeneous Coalition Scheduling with Deadlines . . . 138

6.1 Introduction . . . 138
6.2 Related Work . . . 140
6.2.1 Multi-Robot Scheduling . . . 140
6.2.2 Graph Neural Networks . . . 141
6.3 Problem Description . . . 142
6.4 Building a Heterogeneous Graph . . . 145
6.5 Model . . . 148
6.6 Heterogeneous Temporal Graph Scheduler . . . 151
6.6.1 Greedy Search . . . 151
6.6.2 Beam Search . . . 152
6.7 Experimental Evaluations . . . 153
6.7.1 Ablation Study . . . 154
6.7.2 Comparison with other Scheduling Approaches . . . 155
6.7.3 Usage within the GRSTAPS framework . . . 159
6.8 Discussion & Conclusion . . . 161

Chapter 7: Conclusion and Future Directions 163

7.1 Summary of Contributions 163

7.1.1 Interleaving Task Planning and Scheduling 163

7.1.2 Interleaving Task Allocation, Scheduling, and Motion Planning . . . 164

7.1.3 Interleaving Allocation, Planning, and Scheduling 164

7.1.4 Heterogeneous Coalition Scheduling with Temporal Uncertainty . . 165

7.1.5 Learning to Set Task Orderings for Heterogeneous Coalition Scheduling with Deadlines 165

7.2 Open Questions 165

7.2.1 Reducing Motion Planning Assumptions 166

7.2.2 Uncertainty 167

7.2.3 Execution 169

References 170

LIST OF TABLES

2.1 Number of problems solved for each domains. Red denotes the best value for a row. 27

5.1 Results for different risk levels 127

5.2 Summary of comparison results ($\alpha = 0.1$). Red indicates the best result for the row. 129

5.3 Results for $\bar{p}(C, \rho)$ 132

6.1 Ablation Study of the F1 score for different variants of the HG-GCN architecture. Red: The best model. Violet: The second best model. 155

6.2 Summary of benchmarking results for HCSD Problems. Red: The best value for each metric. Violet: The second best value for each metric. 156

6.3 Summary of benchmarking results for STAP-STC Problems. Red: The best value for each metric. Violet: The second best value for each metric. 160

LIST OF FIGURES

2.1 Caricature of a partial-order plan. 15

2.2 Diagram of the components that each of the task planning layer and scheduling layer contribute to in FCPOP. 20

2.3 $y = 0$ corresponds to the makespan of plans created by FCPOP and points above $y = 0$ are plans created by TFLAP or OPTIC with longer makespans and below are plans they created with shorter makespans. 27

2.4 $y = 0$ corresponds to the planning time of FCPOP and points above $y = 0$ are problems in which TFLAP or OPTIC were slower and below are problems in which they were faster. 28

2.5 Comparison of FCPOP and FCHPOP in terms of plan makespan 30

2.6 Comparision of FCPOP and FCHPOP in terms of planning time 31

2.7 Comparison of FCPOP and FCHPOP in terms of number of nodes visited and explored 31

3.1 High-level architecture of the hierarchical framework. 42

3.2 An example incremental task allocation graph. 43

3.3 The results of ITAGS with various α values normalized with respect to ITAGS$_{\alpha=0.5}$. $y = 0$ corresponds to ITAGS$_{\alpha=0.5}$. Anything above $y = 0$ is worse than ITAGS$_{\alpha=0.5}$ and conversely anything below is better. '*' denotes statistical significance with a p-value < 0.05. 50

3.4 The results of the sequential version of ITAGS (ITAGS$_S$) normalized with respect to ITAGS$_{\alpha=0.5}$. $y = 0$ represents ITAGS$_{\alpha=0.5}$. Anything above $y = 0$ is worse than ITAGS$_{\alpha=0.5}$ and conversely anything below is better. . . 52

3.5 Benchmark against CFLA2 and CCF 54

4.1 Example Task Schemas 68

4.2 An example of $\mathbf{AQ}$ compared to the desired traits matrix $\mathbf{Y}^T$. As $\mathbf{AQ}$ is element-wise greater than or equal to the desired traits matrix, it satisfies the requirements. 71

4.3 An example of the task planning search. The search starts from a plan with a single dummy task and searches to through a plan space to find a solution plan 79

4.4 The associated incremental task allocation graph for plan $\pi_\mathbf{i}$ from Figure 4.3 83

4.5 An example of the scheduling layer's process. Within each box solid lines represent precedence constraints, dashed lines represent mutex constraints, and dotted lines represent the duration of a task. 88

4.6 Example survivor domain map used for the experiments. 92

4.7 High-level architecture of STPA/STAA. 94

4.8 Experiment 1: A comparison of GRSTAPS to both sequential baselines (STPA and STAA) when scaling the number of robots. All problems have 20 goals. 96

4.9 Experiment 2: A comparison of GRSTAPS to both sequential baselines (STPA and STAA) when scaling the number of goals. All problems have 15 agents. 99

4.10 Experiment 3: A comparison of GRSTAPS to three temporal planners (FCPOP, TFLAP, and OPTIC) when scaling the number of robots. All problems have 15 goals. 102

4.11 Experiment 4: A comparison of GRSTAPS to three temporal planners (FCPOP, TFLAP, and OPTIC) when scaling the number of goal. All problems have 15 agents. 105

4.12 Experiment 5: Testing the limits of GRSTAPS using 30 goals and an increasing number of agents. 106

5.1 The blue (red) markers are problem instances where GRSTAPS^{TU} did better (worse) than GRSTAPS^{DET}. Each successive dashed colored line is the result from GRSTAPS^{DET} being 10% worse than the result from GRSTAPS^{TU} (i.e. the first line represents 10% worse, the second 20% worse, etc). 131

6.1 The proposed Heterogeneous Temporal Graph Scheduler approach for solving the Heterogeneous Coalition Scheduling with Deadlines Problem. The Heterogeneous G-GCN block is based on an image from [152] 139

6.2 Example input heterogeneous graph 148

LIST OF ACRONYMS

APR Allocation Percentage Remaining

APSP All-Pairs Shortest Path

BACCHUS Benders Accelerated Cut Creation for Handling Uncertainty in Scheduling

C-Var Conditional Value-At-Risk

CFSTP Coalition Formation with Spatial and Temporal Constraints Problem

CS-HSSRG Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee

CTP Canadian Traveler Problem

DTG Domain Transition Graph

DTP Disjunctive Temporal Problem

EHC Enhanced Hill Climbing

FCHPOP Forward Chaining Hierarchical Partial-Order Planner

FCPOP Forward Chaining Partial-Order Planner

FD Fast Downward

FF Fast Forward

FMAP Forward Multi-Agent Planning

GRSTAPS Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling

HCS Heterogeneous Coalition Scheduling

HCSD Heterogeneous Coalition Scheduling with Deadlines

HCSTU Heterogeneous Coalition Scheduling with Temporal Uncertainty

HG-GCN Heterogeneous Gated Graph Convolution Network

HTGS Heterogeneous Temporal Graph Scheduler

HTNs Hierarchical Task Networks

IPC International Planning Competition

ITAGS Incremental Task Allocation Graph Search

LTL Linear Temporal Logic

MADLA Multi-Agent Distributed and Local Asynchronous

MAFS Multi-Agent Forward Search

MAMP Multi-Agent Motion Planning

MAP Multi-Agent Planning

MAPF Multi-Agent Pathfinding

MAPJA Multi-Agent Planning with Joint Actions

MDP Markov Decision Process

MILP Mixed-Integer Linear Programming

MITL Metric Interval Temporal Logic

MLP Multi-Layer Perceptron

MMMP Multi-Modal Motion Planning

MRC Multi-Robot Coordination

MRSs Multi-Robot Systems

MVR Multi-Vehicle Routing

MVRP-TW Multi-Vehicle Routing Problem with Time Windows

NSQ Normalized Schedule Quality

OMPL Open Motion Planning Library

OPTIC Optimizing Preferences and TIme-dependent Costs

PANDA Planning and Acting in a Network Decomposition Architecture

PARIS Polynomial-time Algorithm for RIsk-aware Scheduling

PDDL Planning Domain Description Language

POCL Partial-Order Causal Linking

POMP Partial-Order Multi-agent Planning

POP Partial-Order Plan

PSM Planning State Machine

PSTP Probabilistic Simple Temporal Problem

PSTPU Probabilistic Simple Temporal Problem with Uncertainty

RCPSP Resource Constrained Project Scheduling Problem

RCPSPU Resource Constrained Project Scheduling Problem with Uncertainty

rrt Rapidly-Exploring Random Trees

SAA Sample Average Approximation

SMT Satisfiability Modulo Theorem

SORU SAA Optimization for solving RCPSP under Uncertainty

SPA Single-agent Planning Approach

SPRT Sequential Probability Ratio Test

STAA Sequential Task Allocation Anytime

STAP Simultaneous Task Allocation and Planning

STAP-STC Simultaneous Task Allocation and Planning with Spatiotemporal Constraints

STN Simple Temporal Network

STP Simple Temporal Problem

STPA Sequential Task Planning Anytime

STPU Simple Temporal Problem with Uncertainty

TAMP Task and Motion Planning

TETAQ Time-Extended Task Allocation Quality

TFLAP Temporal FLAP

TRPG Temporal Relaxed Planning Graph

VHPOP Versatile Heuristic Partial-Order Planning

SUMMARY

In a wide variety of domains, such as warehouse automation, agriculture, defense, and assembly, effective coordination of heterogeneous multi-robot teams is needed to solve complex problems. Effective coordination is predicated on the ability to solve the four fundamentally intertwined questions of coordination: *what* (task planning), *who* (task allocation), *when* (scheduling), and *how* (motion planning). Owing to the complexity of these four questions and their interactions, existing approaches to multi-robot coordination have resorted to defining and solving problems that focus on a subset of the four questions. Notable examples include Task and Motion Planning (*what* and *how*), Multi-Agent Planning (*what* and *who*), and Multi-Agent Path Finding (*who* and *how*). In fact, a holistic problem formulation that fully integrates the four questions lies beyond the scope of prior literature.

This dissertation focuses on **examining the use of shared constraints on tasks and robots to interleave algorithms for task planning, task allocation, scheduling, and motion planning and investigating the hypothesis that a framework that interleaves algorithms to these four sub-problems will lead to solutions with lower makespans, greater computational efficiency, and the ability to solve larger problems.** To support this claim, this dissertation contributes: (*i*) a novel temporal planner that interleaves task planning and scheduling layers, (*ii*) a trait-based time-extended task allocation framework that interleaves task allocation, scheduling, and motion planning, (*iii*) the formulation of holistic heterogeneous multi-robot coordination problem that simultaneously considers all four questions, (*iv*) a framework that interleaves layers for all four questions to solve this holistic heterogeneous multi-robot coordination problem, (*v*) a scheduling algorithm that reasons about temporal uncertainty, provides a theoretical guarantee on risk, and can be utilized within our framework, and (*vi*) a learning-based scheduling algorithm that reasons about deadlines and can be utilized within our framework.

CHAPTER 1
INTRODUCTION

Recent advancements in robotics technology has led to the increased deployment of robotic teams in various domains including agriculture [1], hospitals [2], military [3], warehouses [4], and disaster recovery [5]. This increased deployment is caused by a number of reasons including (*i*) the inherent distribution in space, time, or functionality of many applications, (*ii*) the desire for improved efficiency created by distributing the work load to robots operating in parallel, and (*iii*) the desire for improved robustness and reliability of execution through the redundancy created by the duplication of capabilities across the robot team. Additionally, researchers have begun to focus on coordinating heterogeneous teams of robots because it is cheaper and more practical in many applications to build a number of less capable specialist robots that can work together on a mission, rather than trying to build one generalist robot that can perform the entire mission on its own.

Effective coordination of a robot team is predicated on the ability to solve interacting problems at varying levels of abstraction. Particularly challenging are the four fundamentally intertwined questions of coordination: *what* (task planning), *who* (task allocation), *when* (scheduling), and *how* (motion planning).

While these four questions of coordination have been studied extensively within the context of homogeneous robots (e.g., [6, 7, 8, 9]), these questions require further investigation within the context of heterogeneous robot teams. Recently researchers have started defining and solving problems that focus on some subset of the four questions. Notable examples include Task and Motion Planning (TAMP) [10] (*what* and *how*), Multi-Agent Planning (MAP) [6] (*what* and *who*), and Multi-Vehicle Routing (MVR) [11] (*who* and *how*). However, perhaps due to the complexity of the individual questions and their interactions, existing approaches have not formulated a problem that covers all four questions.

Such a holistic Multi-Robot Coordination (MRC) problem can be quite challenging due to a variety of factors including

- the number of discrete states in the symbolic state space, which grows exponentially as the number of symbolic state variables increases,
- interdependent tasks that can be executed concurrently and require a diverse set of capabilities,
- heterogeneous robots that can form coalitions to execute individual tasks, where the number of possible coalitions grows exponentially as the number of robots and tasks increase,
- long-horizons both in terms of the durations of individual robot paths as well as the overall number of tasks in a solution,
- the dimensionality of the continuous configuration sub-spaces,
- and the interplay of discrete and continuous decisions.

The most straightforward approach for attempting to solve one of these holistic MRC problems is to reduce it to a constrained mathematical program and solve for values for all of the free parameters at once. Although there is a vast literature on mathematical programming and highly efficient commercial numerical solvers exist, this ends up being infeasible in practice for all but the smallest of problems due to the large number of combinatoric factors (e.g., number of tasks, number of robots, number of objects in the environment, size of the environment, etc).

An alternative approach would be to chain together a series of existing state-of-the-art algorithms for each of the sub-problems. However, when exploring subsets of the problem, previous research has found that this approach is unable to solve the general class of problems [12]. This is because the interdependence of the sub-problems is an essential component of the overall problem. A common example from the TAMP literature that demonstrates this concept is a robot that wants to stack two cups on a shelf. If it only considers the task planning portion of the problem, it may create a symbolic plan where it puts

the first cup on the shelf and then attempts to put the second cup on top of the first. It is possible that the robot cannot reach high enough to put the second cup on top of the first, but if it had considered the geometric information, it would have stacked the cups and then moved the entire stack to the shelf.

This dissertation examines the formulation of a class of holistic heterogeneous multi-robot coordination problems which we have named Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC). Additionally, it explores the concept that while algorithms for each of the four sub-problems utilize different properties of the shared components, namely tasks and robots, they can benefit from a shared collection of constraints to interleave information that each algorithm discovers while solving its sub-problem. We posit that utilizing these shared constraints allow for interleaved frameworks with improved computation efficiency that can scale to larger problems[1]. As part of this examination, we first develop interleaved frameworks that exploit this concept for two subsets of the STAP-STC problem in Temporal Planning (*what* and *when*) and Time-Extended Task Allocation (*who*, *when*, and *how*). We then develop a unified and interleaved framework that passes shared constraints between four modules, one for each sub-problem, to solve the STAP-STC problem. We evaluate the performance of interleaving algorithms for each of these frameworks through ablation studies against sequential variants of the frameworks and comparisons against state-of-the-art algorithms. In these evaluations, we consider solution makespan, computation time, and success rate relative to problem size.

1.1 Thesis Statement

This thesis **examines the use of shared constraints on tasks and robots to interleave algorithms for task planning, task allocation, scheduling, and motion planning and investigates the hypothesis that a framework that interleaves algorithms to these four sub-problems will lead to solutions with lower makespans, greater computational ef-**

[1]For this dissertation, we measure the scale of a problem in terms of number of robots, number of tasks, or number of symbolic goals.

ficiency, and the ability to solve larger problems.

1.2 Contributions

To support this claim, this dissertation makes the following contributions to the field of heterogeneous multi-robot coordination:

- **Interleaving Task Planning and Scheduling**: (Chapter 2) We introduce a novel temporal planner, Forward Chaining Partial-Order Planner (FCPOP), which interleaves algorithms for task planning and scheduling. FCPOP utilizes techniques from both state-space-based forward chaining and plan-space-based partial-order planning. We also build upon FCPOP to introduce a novel hierarchical temporal planner, Forward Chaining Hierarchical Partial-Order Planner (FCHPOP). FCHPOP integrates techniques from Hierarchical Task Networks through hierarchical abstraction on tasks. We evaluate FCPOP through comparisons against state-of-the-art temporal planners from the last International Planning Competition's Temporal Track. Through these experiments we demonstrate that FCPOP improves the efficiency of solving temporal planning problems while on average producing higher quality solutions over state-of-the-art temporal planners. Last, we perform an ablation study to demonstrate the advantages of utilizing abstract tasks through a comparison of FCHPOP and FCPOP. We demonstrate that on average FCHPOP explores fewer nodes which leads to a reduced computation time and it utilizes the domain knowledge provided in the abstract tasks to produce higher quality solutions.
- **Interleaving Task Allocation, Scheduling, and Motion Planning**: (Chapter 3) We develop a novel hierarchical framework that interleaves task allocation, scheduling, and motion planning to simultaneously address coalition formation and scheduling for heterogeneous multi-robot systems. Additionally, we introduce a search-based approach for trait-based task allocation, named Incremental Task Allocation Graph Search (ITAGS). ITAGS leverages two complementary heuristics in Allocation Per-

centage Remaining (APR) and Normalized Schedule Quality (NSQ) and a convex combination of the two heuristics in Time-Extended Task Allocation Quality (TETAQ). We evaluate the relative influence of each of the two heuristics on the performance of ITAGS. We then perform an ablation study where we study the benefits of interleaving the information through shared constraints between sub-problem modules. Finally, we demonstrate the efficacy and solution quality of ITAGS through a comparison against two state-of-the-art Task Allocation algorithms.

- **Interleaving Allocation, Planning, and Scheduling**: (Chapter 4) We formulate a holistic heterogeneous multi-robot coordination problem, named Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC), that simultaneously considers the four questions of coordination: *what* (task planning), *who* (task allocation), *when* (scheduling), *how* (motion planning). We introduce an extension to our previous hierarchical framework, ITAGS, through the addition of a task planning layer based on FCPOP. The new hierarchical framework, named Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling (GRSTAPS), is our initial solution to the STAP-STC problem. We perform an ablation study where we study the benefits of interleaving the information through shared constraints from all four sub-problem modules on several problems that scale in terms of number of symbolic goals and number of robots. We then compare the performance of GRSTAPS against three state-of-the-art temporal planners on more problems that scale in terms of number of symbolic goals and number of robots. Finally, we test the limits of how large of a problem can GRSTAPS solve within a specified time limit.
- **Heterogeneous Coalition Scheduling with Temporal Uncertainty**: (Chapter 5) We introduce a novel class of uncertainty-aware scheduling problem, the Heterogeneous Coalition Scheduling with Temporal Uncertainty (HCSTU) problem, which explicitly considers the uncertainties in the time needed to execute a task and to transition between two tasks. To solve the HCSTU, we present a sampling-based risk-

aware algorithm named Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee (CS-HSSRG). CS-HSSRG provides theoretical guarantees on a user provided acceptable amount of risk through utilization of the Sequential Probability Ratio Test [13] and is demonstrated to be extremely efficient when compared to state-of-the-art approaches, while not being overly conservative. Additionally, we test CS-HSSRG as the scheduling component of GRSTAPS and show that utilizing CS-HSSRG aids GRSTAPS in making robust decisions about what tasks to include in the task plan and what robots to allocate to those tasks.

- **Learning to Set Task Orderings for Heterogeneous Coalition Scheduling with Deadlines**: (Chapter 6) We introduce a Heterogeneous Gated Graph Convolution Network (HG-GCN) model that learns the probability that pairs of tasks are ordered in a specific way in an optimal schedule. This model takes as input a heterogeneous graph that represents the relationships between time points and can encode all 13 of Allen's temporal relationships [14]. Furthermore, we present a search-based approach named Heterogeneous Temporal Graph Scheduler (HTGS) with two search strategies that utilizes the output of the HG-GCN model to approximately solve the Heterogeneous Coalition Scheduling with Deadlines problem. We evaluate the performance of this approach through a comparison against state-of-the-art heterogeneous multi-robot scheduling algorithms and an optimal Mixed-Integer Linear Programming baseline. Additionally, we integrate the approach into GRSTAPS and evaluate the improvement in performance of the overall framework. The results of these evaluations demonstrate the efficacy of the new approach.

1.3 Relevant Publications

A. Messing and S. Hutchinson, "Learning to Set Task Orderings for Heterogeneous Coalition Scheduling with Deadlines," IEEE Transactions on Robotics, 2023.

A. Messing[*], J. Banfi[*], M. Stadler, E. Stump, H. Ravichandar, N. Roy, and S. Hutchinson,

"Heterogeneous Coalition Scheduling with Temporal Uncertainty," IEEE Transactions on Robotics, 2023.

J. Banfi*, **A. Messing***, C. Kroninger, E. Stump, S. Hutchinson, and N. Roy, "Hierarchical Planning for Heterogeneous Multi-Robot Routing Problems via Learned Subteam Performance," IEEE Robotics and Automation Letters, vol. 7, no. 2, pp. 4464-4471, 2022.

A. Messing*, G. Neville*, S. Chernova, S. Hutchinson, and H. Ravichandar, "Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling," The International Journal of Robotics Research, vol. 41, no. 2, pp. 232-256, 2022.

G. Neville*, **A. Messing***, H. Ravichandar, S. Hutchinson, and S. Chernova, "An Interleaved Approach to Trait-based Task Allocation and Scheduling," International Conference on Intelligent Robots and Systems, 2021.

A. Messing and S. Hutchinson, "Forward Chaining Hierarchical Partial-Order Planning," International Workshop on the Algorithmic Foundations of Robotics, vol. 14, pp. 364-380, 2020.

1.4 Dissertation Organization

The rest of this dissertation is organized as follows. Chapter 2 introduces temporal planning as a mixture of task planning and scheduling and focuses on the Forward Chaining Partial-Order Planner and the Forward Chaining Hierarchical Partial-Order Planner. Chapter 3 transitions to focus on time-extended task allocation by interleaving task allocation, scheduling, and motion planning through the Incremental Task Allocation Graph Search. Chapter 4 then formalizes a holistic heterogeneous multi-robot coordination problem in Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) and presents an initial approach to solve the STAP-STC Problem in Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling. Chapter 5 introduces the Heterogeneous Coalition Scheduling with Temporal Uncertainty problem and presents Coalition

*Co-First Authors

Scheduling with Heuristic Sample Selection and Risk Guarantee as an efficient risk-aware approach with theoretical guarantees. Chapter 6 discusses a Heterogeneous Gated Graph Convolution Network (HG-GCN) model that learns the probability that pairs of tasks are ordered a specific way in the optimal solution and Heterogeneous Temporal Graph Scheduler as a search-based approach that utilizes the output of the HG-GCN model to solve the Heterogeneous Coalition Scheduling with Deadlines problem. Finally, we provide concluding remarks and discuss open questions in Chapter 7.

CHAPTER 2

INTERLEAVING TASK PLANNING AND SCHEDULING

2.1 Introduction

Prior to focusing on the full Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) problem that we mentioned in Chapter 1, in this chapter, we first focus on interleaving Task Planning and Scheduling for a problem known as Temporal Planning[1]. Temporal planning provides the tools to enable intelligent behaviors in robotic systems operating in the real world while considering temporal constraints and concurrent tasks. The need for efficient algorithms that allow for these intelligent behaviors has led to a resurgence of Temporal Planning research in the robotics community.

Partial-order planning (POP) was the most popular method for classical task planning until the late 90s. With its least-commitment philosophy, POP has several important advantages, such as being more flexible in execution [15], easy to extend for temporal planning [16], very suitable for multi-agent planning systems [17], and similar structure to temporal networks [18]. The flexibility of these plans makes them easier to execute and to repair for real-world systems, as small delays in a robot's tasks can be dealt with without modifying the plan. Also, the partial-ordering creates a useful model for concurrent execution, which is useful for a single robot with multiple sub-systems and for multi-robot systems. However, POP is known to be slower than other modern planning methods that use improved solvers, more informed state-based heuristics, and lighter-weight search machinery that allows for much faster backtracking across alternative search states. The speed deficiency led to it becoming unfashionable in the task planning community [18].

[1]The material in this chapter is based on:
A. Messing and S. Hutchinson, "Forward Chaining Hierarchical Partial-Order Planning," International Workshop on the Algorithmic Foundations of Robotics, vol. 14, pp. 364-380, 2020.

The desire for a temporal planning algorithm that has the flexibility of POP and the speed of modern day planning algorithms has led to recent investigation into combining POP with other planning techniques. One example of this is combining POP with grounded forward chaining to gain the speed of better state-based heuristics and backtracking mechanisms [18]. This requires a relaxation of the least-commitment philosophy that POP is built on. Another approach that has been explored is combining hierarchical information, similar to what is used in a Hierarchical Task Networks (HTNs), with POP in what is known as hybrid planning [19].

This chapter introduces the Forward Chaining Partial-Order Planner (FCPOP), a temporal planner that interleaves shared constraints between causality-based task planning and temporal network-based scheduling. Furthermore, FCPOP combines partial-order plan construction with a grounded forward chaining search, however, unlike other approaches that combine the two techniques, FCPOP utilizes the full delayed task ordering to create more flexible, higher quality plans with shorter makespans and that are suitable for multi-robot systems. The utilization of delayed task ordering also reduces and sometimes eliminates the need for backtracking which causes our planner to be more efficient than many state-of-the-art temporal planners. In addition, through grounded forward chaining FCPOP can use a temporal state-based heuristic for a more informed search through the plan space and avoids deadends that might arise due to temporal constraints. Despite the extra computation cost from the more informed heuristic and the full delayed task ordering, we empirically demonstrate that FCPOP outperforms state-of-the-art temporal planners on several benchmarks from the most recent temporal track of the International Planning Competition [20].

Additionally, this chapter introduces the Forward Chaining Hierarchical Partial-Order Planner (FCHPOP), which builds upon FCPOP. FCHPOP includes hierarchical information in the form of abstract tasks that can be recursively refined. The hierarchical structure of tasks allows expert knowledge about a domain (e.g, a predefined procedure) to be in-

cluded with relative ease to guide the search process an speed up planning overall [21]. The task hierarchy can be built so the planning algorithm can plan for teams instead of individual robots. This allows for solutions where robots on the same team can shared the same high-level goal and act together as much as possible without the need to plan each task for each individual robot separately [19]. As such, planning with these abstract tasks and refining them reduces the number of nodes that need to visited and explored during search which speeds up the process while still creating a quality plan with a low makespan. This is empirically demonstrated through an ablation study against FCPOP on multiple benchmark problems.

2.2 Background

In this section, we give an overview of the background information for the relevant task planning problems. Before we go into the details for each of these select task planning problems, we look at the common components. Most task planning approaches view the world as a state transition system and the role of the task planner is to select and organize tasks[2], which work as transition functions, to change the state of the system. The description of features of the state and the tasks that can change the state are given by a task planning domain model, which typically consists of a formal language and *operator* descriptions. For this thesis, we use Planning Domain Description Language (PDDL) [22] as it is the current standard language for the International Planning Competition [23, 20]. An *operator* is a parameterized task schema that represents the preconditions and effects through logical formulas. When the parameters of an operator become grounded, then it becomes a task. A *plan* in this context is a set of tasks and a set of constraints on the precedence/ordering of these tasks, however, there are more sophisticated representations.

A *task planning problem* is typically specified by a domain model, an initial state, and

[2]We use the term task because that is the term used in the multi-robot literature and here to maintain consistency throughout this document. A task is synonymous with the term action as used in the AI planning literature.

a set of conditions that must be satisfied to reach the goal. A plan is considered to be a *solution* to a task planning problem if it is constructed from instances of operators from the given domain model, executable from the initial state, and its execution results in a state that satisfies the conditions of the goal.

Implementations of this paradigm have been employed successfully in many application areas and most importantly for our use case, in many robotic application areas such as assembly [24], warehouse automation [25], hospitals [26] and in the home [27].

2.2.1 Task Planning

In task planning, an agent/planner chooses a set of tasks such that when these tasks are executed in order from an initial state the world is transformed into a state that satisfies the conditions of the goal. Typically, the world is described by a set of *state variables* $V = \{v_0, \ \ldots, \ v_k\}$ that depict high-level properties of physical objects (e.g. that a robot is holding a box, rather than the actual coordinates of the robot and the box) or relations between objects (e.g. whether a cup is on top of a table). Each state variable, v, is associated to a finite domain, D_v, of mutually exclusive values that refer to objects of the world.

Each *task* τ, which represents a state transition function, contains a set of simple conditions and a set of simple effects. A *simple condition* $c \equiv [v = x]$ is an assertion that a state variable v has the value $x \in D_v$. A *partial state* $\grave{s} = \{c_0, \ \ldots, \ c_m\}$ is a set of simple conditions. If the number of simple conditions in the partial state is not the same as the total number of state variables ($|\grave{s}| \neq |V|$) then the partial state represents multiple possible states. A *state* s is a set of simple conditions where the number of simple conditions is equal to the number of state variables ($|s| = |V|$). A *simple effect* $e \equiv v \leftarrow x$ is an assignment of the value $x \in D_v$ to the state variable v.

The task planning problem can be represented as a tuple $\langle V, \mathcal{T}, s_{init}, G \rangle$ where

- V is a set of state variables,
- $\mathcal{T}$ is a set of tasks,

- s_{init} is the initial state,
- G is a set of simple conditions.

Modern task planners usually attempt to minimize the number of tasks in the plan or associate costs to tasks and attempt to find a solution with the minimum total cost. There are two main approaches to solve the task planning problem: total-order planning and partial-order planning.

Total-Order Planning

In total-order planning, also known as linear planning, an agent/planner chooses a sequence of tasks such that when these tasks are executed one at a time in order from an initial state the world is transformed into a state that satisfies the conditions of the goal This sequence of tasks is known as totally-ordered plan. The fastest task planners today are linear planners that conduct a forward search through a space of states, commonly known as forward chaining or forward state-progression, and utilize relaxed problem specifications as a heuristic function for search control [28].

Partial-Order Planning

Partial-order planning, also known as non-linear planning or Partial-Order Causal Linking (POCL), started out as an alternative to linear/total-order planning approaches. This was mainly motivated by the desire to find solutions for problems with interacting sub-goals, such as the Sussman Anomaly [29]. For task planning problems, the community has since gone back to linear planning approaches due to improvements in search speed and more informed heuristics, however, partial-order planning has recently seen a resurgence in the temporal and multi-agent planning community due to its flexibility and ability to model task concurrency (discussed more in detail down below).

Traditional partial-order planning is a planning approach that uses a least-commitment philosophy where it postpones decisions about the order of task and parameter bindings

until a decision is forced. In postponing the decision about task orderings, only the essential ordering decisions are stored and plans are represented as a partially-ordered set of tasks instead of committing prematurely to a totally-ordered sequence of tasks. Parameter binding decisions are also delayed so that task parameters are only grounded as required for causal links [30].

A *partial-order plan* is represented as a tuple $\pi = \{\mathrm{T}, \mathcal{L}, \mathcal{P}\}$ where $\mathrm{T} \subseteq \mathcal{T}$ is a set of tasks, $\mathcal{L}$ is the set of causal links, and $\mathcal{P}$ is the set of precedence constraints ($\prec$) on T. A *causal link* is a relationship between two tasks, τ_i and τ_j, meaning that a simple condition $c_{v_k} \equiv [v_k = x]$ of τ_j is supported by a simple effect $e_{v_k} \equiv v_k \leftarrow x$ of τ_i. The causal link is represented by $\tau_i \xrightarrow{v_k} \tau_j$ and creates the constraint that τ_i must complete before τ_j can start. A *precedence constraint* $\tau_i \prec \tau_j$ is a temporal relationship between two tasks, τ_i and τ_j, that enforces that τ_i must complete before τ_j starts. When considering the ordering of tasks, a causal link is specific type of precedence constraint. Each partial-order plan represents a set of one or more total-order plans.

Partial-order planners search through a space of plans, where each node in the search represents a partial-order plan and each edge represents plan refinement operations such as adding a task [30]. The planner starts by creating a fictitious task τ_G whose conditions are the conditions of G and then creates a root node where the partial-order plan it represents contains only τ_G ($\pi_0 = \{\{\tau_G\}, \emptyset, \emptyset\}$). The planner also creates a fictitious task τ_{init} whose effects are the simple effects needed to create the initial state s_{init}.

At each stage of planning, a partial-order planner chooses a node in the search tree and determines what flaws that node has. A *flaw* is something that prevents a partial-order plan from being a solution to the planning problem.

The first type of flaw is an *open link* where one of the simple conditions c_{v_k} from a task τ_j in the partial-order plan does not have a causal link to support it. This can be resolved by creating a causal link from a task τ_i to τ_j over the state variable v_k ($\tau_i \xrightarrow{v_k} \tau_j$). The task τ_i can either already be in the plan ($\tau_i \in \mathrm{T}$) or be a new task that is added ($\tau_i \notin \mathrm{T}; \tau_i \in \mathcal{T}$).

The second type of flaw is a *threat*, which is a conflict between a task τ_k added to the plan and a causal link ($\tau_i \xrightarrow{v_y} \tau_j$). This conflict is created when a simple effect of the task τ_k modifies a state variable away from the value needed for the causal link and there is no temporal relationship between τ_k and either τ_i or τ_j that would prevent it from doing so. A threat from task τ_k on causal link $\tau_i \xrightarrow{v_y} \tau_j$ can be resolved through either adding a precedence constraint $\tau_k \prec \tau_i$ known as a promotion or $\tau_j \prec \tau_k$ known as a demotion.

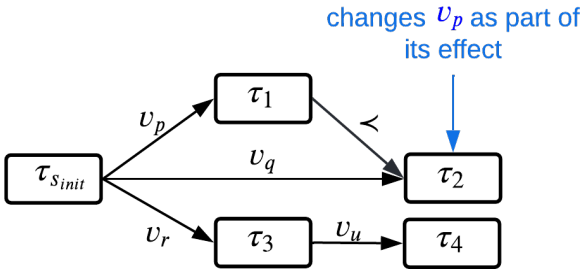


Figure 2.1: Caricature of a partial-order plan.

We show a caricature of an example partial-order plan in Figure 2.1. In this example partial-order plan, four tasks have been added through four causal links, however τ_2 changes v_p as one of its effects. As the causal link between $\tau_{s_{init}}$ and τ_1 is over v_p this creates a threat on that causal link. As such, τ_2 was promoted through the precedence constraint $\tau_1 \prec \tau_2$ which resolves the threat and creates a flaw free partial-order plan.

Partial-order planners search by heuristically determining which node in the search tree to expand, which flaw to resolve for that node, and then how to resolve the selected flaw. The result of the resolution creates a new node in the tree. A solution is found a partial-order plan is created that contains τ_{init} and there are no remaining flaws. This is, all tasks' conditions have been satisfied through causal links and there are no remaining threats. As such, partial-order planning performs a plan-based, backward search, refining partial-order plans through the addition of tasks, causal links, and precedence constraints.

2.2.2 Scheduling

Scheduling is the process of reasoning about how a given set of tasks with temporal constraints can be executed in a limited or minimized amount of time. As such, the objective of a scheduler is generate a *schedule* as an assignment of start and completion time points for each task while taking into account any deadlines or predefined orderings of tasks. The main conceptual difference between planning and scheduling is that for scheduling the set of tasks are known in advance whereas planning determines what tasks should be executed based on the causality of conditions and effects.

2.2.3 Temporal Planning

Temporal planning integrates task planning with scheduling. The temporal planning problem utilizes the same problem input as task planning with the exception that tasks also contain a predefined duration and are known as durative tasks. Additionally, temporal planning introduces added complexity as durative tasks can be executed concurrently as long as there are no causality-based ordering between them. The solution to a temporal planning problem is a plan as a set of tasks and a schedule. This naturally lends itself to partial-order planning and led to a number of temporal planners that integrate a partial-order planner with a scheduler.

2.2.4 Hierarchical Planning

The two primary issues of the approaches to the planning problems discussed so far are scalability and the lack of ability to incorporate domain expert procedural knowledge. Planning problems are NP-Hard and quickly escalate in difficulty as more tasks and state properties are defined. On the other hand, while domain expertise can be used in heuristics for selecting single tasks or orderings under the previous problems there is no way to utilize domain expertise to include multiple tasks with previously known causality and temporal relationships (i.e. a known procedure). These observations motivated the use of abstrac-

tion mechanisms in planning, which is usually divided into two categories: approaches that utilize abstraction for the representation and reasoning about states, and approaches that utilize abstract hierarchies on tasks [28]. We will focus on the problem when abstractions are utilized for hierarchies on tasks. More details about approaches that utilize abstraction with states can be found in [28] and [31].

The idea behind utilizing hierarchical abstraction tasks comes from a simple question: why synthesize the tasks to be used within a plan one at a time if there is available knowledge (e.g., recorded domain expertise) on how to achieve one or more of the conditions in the goal. This leads to the concepts of abstract tasks and primitive tasks. A *primitive task* is the same as the tasks that we have been discussing up until now. An *abstract task* τ in addition to the features of a primitive task (e.g., conditions, effects, duration) also has a set of methods. A *method* contains either a totally-ordered or partially-ordered plan that must be supported by the conditions of τ and when executed achieves the effects of τ. A method can contain both abstract tasks and primitive tasks. Converting an abstract task to one of its methods is called decomposition.

Instead of starting with an initial state and trying to find a sequence/set of tasks that when executed satisfy the conditions of a goal like the previous planning problems, hierarchical planning starts with an initial plan with one or more abstract tasks. The job of a hierarchical planner is to recursively decompose the abstract tasks in the plan by selecting which method to use for each until there are only primitive tasks in the plan. This approach to planning has been subsumed under the name Hierarchical Task Networks (HTNs) planning.

2.2.5 Multi-Agent Planning

While task planning usually considers just a singular agent, multi-agent planning focuses on synthesizing a plan when there are multiple agents that can participate in the plan. As such, multi-agent planning integrates elements from task planning, task allocation, and

scheduling. For our purposes, it should be noted that a multi-agent planning problem can be cast as a temporal planning problem by including features of the agents as state variables. Further information about multi-agent planning can be found in [6].

2.3 Related Work

Due to better modeling of real-world problems which typically involve both task planning and scheduling, in recent years temporal planning has become a more central interest of the planning community.

Younes and Simmons [32] developed Versatile Heuristic Partial-Order Planning (VHPOP) which is able to utilize durative tasks within partial-order planning and utilizes a Simple Temporal Network (STN) [33] to reason about temporal information. However, VHPOP suffers from some of the problems encountered in earlier partial-order planners and its performance scales poorly in many domains.

Schattenberg *et al.* [28] present Planning and Acting in a Network Decomposition Architecture (PANDA) as an architecture for planning and scheduling. Panda utilizes a lifted hierarchical planner while reasoning in plan space. Panda only supports limited qualitative time and has been demonstrated to have limited scalability [34].

Eyerich *et al.* [35] extended the popular classical planner Fast Downward [36] to search through a space of time-stamped states and splits tasks into instantaneous start tasks and instantaneous compressed tasks which are sequenced.

Coles *et al.* developed POPF [18] as a temporal planner that uses a partial-order plan construction within a forward-chaining framework, working with time, numbers, and continuous effects. POPF creates a frontier state at each step of the plan to determine the applicable tasks and relaxes the least-commitment of task order decisions to a late-commitment approach where newly applied tasks cannot come before the frontier state. They ensure completeness as the search backtracks to find an alternative plan when necessary.

Benton *et al.* developed Optimizing Preferences and TIme-dependent Costs (OPTIC) [16]

as an extension of POPF that handles soft constraints and preferences. OPTIC has been demonstrated to be one of the most effective planners in many domains due to its speed at generating successor states during search and the use of effective temporal domain-independent heuristics [37]. It was used as the baseline approach for the most recent temporal track of the International Planning Competition (IPC) [20].

Sapena *et al.* developed TFLAP [37, 15] as temporal planner that similarly uses a partial-order plan construction within a forward-chaining framework, but unlike POPF/OPTIC TFLAP does not require the frontier state to determine the tasks that can be applied to a plan. This fully utilizes the least-commitment approach for ordering tasks and reduces the need for backtracking when building a solution. TFLAP uses a combination of a Domain Transition Graph (DTG) [36], the traditional Fast Forward (FF) heuristic h_{FF} [38], and a landmark graph-based heuristic for guiding its search, however it doesn't use any temporal information in any of its heuristics.

2.4 Forward Chaining Partial-Order Planner

As temporal planning has become a more central interest of the planning community, investigation began into ways to improve the speed of POP due to its abilities to create plans that are more temporally flexible and to better handle concurrent tasks than state-based planners. Combining POP with grounded forward chaining has shown to maintain some of the benefits of POP while exploiting the speed and informed state heuristics of forward chaining.

Forward Chaining Partial-Order Planner (FCPOP) is our variant of combing forward chaining with partial-order planning. FCPOP removes the delayed parameter binding of POP by grounding the operator descriptions into parameterized instances known as tasks before search, but fully utilizes the delayed task ordering commitment from POP and does not create a frontier state for determining which tasks can be added to a plan. This allows tasks to be added to any point of the plan and reduces the amount of backtracking needed

during search. As such, the forward search can be viewed as a commitment to a set of tasks and not to the order of their application [15]. Furthermore, it interleaves algorithms for task planning and scheduling with each affecting different aspects of the search (See Figure 2.2).

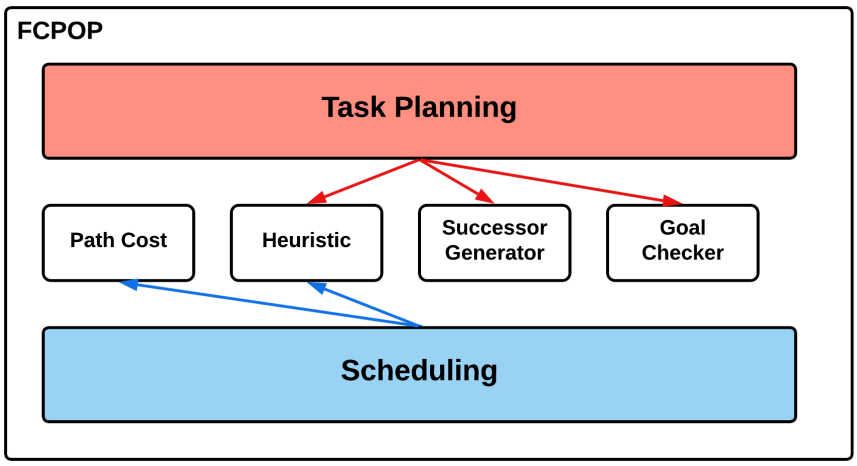


Figure 2.2: Diagram of the components that each of the task planning layer and scheduling layer contribute to in FCPOP.

FCPOP implements an A* search through plan space where each node of the tree is a partial-order plan. For each timed-initial-literal, a fictitious task τ_{til} is created. This task starts at time 0 and lasts until the time at which the timed-initial-literal would happen. The task has no preconditions and a single effect for setting the specific fluent from the timed-initial-literal. FCPOP also creates fictitious tasks for the initial state (τ_{init}) and the goal conditions (τ_G). The search starts at a the root node which contains a partial-order plan $\pi_0 = \langle\{\tau_{init}\}, \emptyset, \emptyset\rangle$ with the fictitious task τ_{init} and no causal links or precedence constraints. The processes of expanding a node for FCPOP is shown in Algorithm 1.

During the expansion of a node π, the fictitious task τ_G is checked for whether it can be added to the partial-order plan from the expanding node. If causal links can be created from the tasks currently in π to support the preconditions of τ_G then a new partial-order plan is created for each possible combination of causal links that supports τ_G. Each of

Algorithm 1: FCPOP Node Expansion

Input: π, The node to expand; $\mathcal{T}$, The set of grounded tasks

if *causal links can be created from the actions in* π *to support task* τ_G **then**
  $P \leftarrow$ list of partial-order plans from adding τ_G to π using Algorithm 2
  if $|P| > 0$ **then**
    $\pi_{best} \leftarrow NULL$
    $C_{best} \leftarrow \infty$
    foreach *partial-order plan* $\pi' \in P$ **do**
      if *a temporally consistent STN can be computed from* π' **then**
        $\sigma' \leftarrow$ the consistent STN computed from π'
        $C' \leftarrow$ the time taken to execute the plan as computed by σ'
        if $C' < C_{best}$ **then**
          $\pi_{best} \leftarrow \pi'$
          $C_{best} \leftarrow C'$
    if $\pi_{best} \neq NULL$ **then**
      π_{best} is the solution to the planning problem
for *task* $\tau \in \mathcal{T}$ **do**
  $P \leftarrow$ a list of partial-order plans from adding τ to π through Algorithm 2
  foreach *partial-order plan* $\pi' \in P$ **do**
    add π' as child node of π'

the partial-order plans is converted to a Simple Temporal Network (STN) [33] which will be explained below. From the STN, the fastest schedule that can be created from the new partial-order plan is calculated. The partial-order plan that can create the fastest schedule and now contains τ_G is the solution.

If τ_G cannot be added to the partial-order plan of π then FCPOP determines which tasks can be added. a task is only feasible to add if causal links can be created from the tasks currently in π to support the new task's preconditions. It is possible there are more than one combination of causal links that support adding the new task. For each of these combinations, the planner determines what threats are created by adding the task, and then if all of the threats are resolvable through precedence constraints. For each combination where all threats are resolvable, a new child node is created with the addition of the new task, the specific combination of causal links, and the precedence constraints needed to

Algorithm 2: FCPOP Adding a Task to a Partial-Order Plan

Input: π, A partial-order plan; τ, A durative task
Output: P, A list of partial-order plans

```
P ← []
if causal links can be created from the tasks in π to support τ then
    L ← a list of the causal link combinations for adding τ to π
    foreach causal link combination l ∈ L do
        T ←a list of the threats from adding τ and l to π
        P ← []
        foreach threat t ∈ T do
            if t can be resolved through an ordering constraint then
                add the precedence constraint that resolves t to P
            else
                break
        if All threats were resolvable then
            π' ← copy of π
            add τ, l, P to π'
            add π' to P;
return P
```

resolve the resulting threats[3].

The process of adding a task to a partial-order plan is shown in Algorithm 2. This process uses elements of POP in adding causal links and resolving threats through precedence constraints; however, it performs forward chaining instead of backwards reasoning. Also, each node in the search tree, unlike traditional POP, is a flaw-free partial-order plan.

For each new node a Simple Temporal Network (STN) [33], as a labeled directed graph, is created. In creating this STN, each durative task from the current partial-order plan is split into two instantaneous tasks representing the start and end of the task. A vertex is created for each of these instantaneous tasks. Edges are added between the start and end instantaneous tasks from the same durative task for the lower and upper bounds on the task's duration. Edges are also added between an end instantaneous task and a start instantaneous task if there is an precedence constraint or causal link between them. As such, the partial-

[3]Empirically, for most planning problems there are not a lot of different causal link combination for adding a single task, however for problems where the number of combination reduces efficiency, FCPOP can be set to only add the X best causal link combinations with X being a user defined parameter.

order plan and the STN shared the constraints between tasks.

Computing a shortest path from the vertex representing τ_{init} identifies the earliest possible time at which the instantaneous task could occur. This can be used to determine a timestamp for each of the instantaneous tasks and to create a schedule. If a negative cycle is found in the STN then the plan is temporally inconsistent, meaning that a task would have be executed before itself. In this case, the cost of the node is set to infinite and a heuristic is not calculate, so that the node is never expanded. If the plan is consistent then the path cost of the node is the total time taken by executing the schedule. Simulating the effects of the tasks in the schedule creates a frontier state which can be used for heuristic evaluation.

The frontier state is used as the initial state for a temporal state-based heuristic. FCPOP uses a modified variant of the Temporal Relaxed Planning Graph (TRPG) from [16] in which layers can have fluents that contain values from finite domains of mutually exclusive values. For these fluents, the label contains timestamps of the transitions between values and each fact is assumed to simultaneously be all achieved values. This still relaxes the planning and the heuristic results in a relaxed plan. The amount of time need to execute the relaxed plan is computed and is used as the heuristic value for the current node.

During the search process it is possible for a node to be generated that has the same plan as a node already in the tree which is a common problem in most forward-search planners [15]. FCPOP uses a memoization technique to avoid this issue. As all the successors are generated for each expanded node, FCPOP's planning algorithm is sound and complete.

2.5 Forward Chaining Hierarchical Partial-Order Planner

While task and temporal planner focus on achieving a goal, hierarchical planners focus on taking an initial plan of abstract tasks or tasks and decomposing them into less abstract tasks until create a plan of tasks that can no longer be decomposed. As part of the plan-

Algorithm 3: FCHPOP Node Expansion

Input: π, The node to expand; $\mathcal{T}_p$, The set of grounded primitive tasks; $\mathcal{T}_a$, The set of grounded abstract tasks

```
1 Add primitive tasks using Algorithm 1
2 Add abstract tasks as if they were primitive tasks using Algorithm 1
3 T_π ← the abstract tasks in π
4 for abstract task τ_a ∈ T_π do
5     M ← the methods of τ_a
6     foreach method m ∈ M do
7         P ← a list of partial-order plans by adding m to π using Algorithm 4
8         foreach partial-order plan π' ∈ P do
9             add π' as a child node of π
```

ning process the planners are provided with plan fragments, also known as methods, that are intended to implement a specific abstract task. These methods usually also contain further abstract tasks, so substitutions of methods for abstract tasks has to be performed recursively. This substitution step is called a decomposition of the abstract task and creates a corresponding hierarchy on the tasks such that tasks in the method are considered less abstract than the substituted abstrtask task. When a task cannot be decomposed any further it is called a primitive task, which corresponds to the task from task or temporal planning. Generally there are more than one possible methods for an abstract task. Over time, this form of planning has been subsumed under the label Hierarchical Task Networks (HTNs) planning [28].

Forward Chaining Hierarchical Partial-Order Planner (FCHPOP) extends FCPOP to use hierarchical information. Previously described durative tasks will be known as primitive tasks for FCHPOP. FCHPOP also uses higher-level tasks known as abstract tasks which can refined into a partial-order plan of tasks (both primitive and abstract).

FCHPOP, as with FCPOP, builds a tree of nodes that contain partial-order plans. The main difference is when FCHPOP expands a node, in addition to adding primitive tasks to the partial-order plan as in FCPOP, it can add the highest-level abstract tasks to partial-order plans, and decompose already included abstract tasks as shown in Algorithm 3. FCH-

Algorithm 4: FCHPOP Adding the Method from an Abstract Task to a Partial-Order Plan

Input: π, A partial-order plan; m, A method from an abstract task
Output: P, A list of partial-order plans

$\pi_m \leftarrow$ the partial-order plan from m
$\mathcal{T}_m \leftarrow$ the list of tasks (both abstract and primitive) from π_m
$\mathcal{L}_m \leftarrow$ the list of causal links from π_m
$\mathcal{P}_m \leftarrow$ the list of precedence constraints from π_m
$\pi' \leftarrow$ copy of π
add $\mathcal{L}_m$ and $\mathcal{P}_m$ to π'
sort $\mathcal{T}_m$ based on $\mathcal{P}_m$ and $\mathcal{L}_m$
$P \leftarrow [\pi']$
for *task* $\tau \in \mathcal{T}_m$ **do**
    $L_\tau \leftarrow$ a list of combinations of causal links to support the unsupported preconditions of τ
    $P_\tau \leftarrow []$
    foreach *partial-order plan* $\pi' \in P$ **do**
        foreach *causal link combination* $l \in L_a$ **do**
            $T_l \leftarrow$ a list of the threats from adding τ and l to π'
            $\mathcal{P}_l \leftarrow []$
            foreach *threat* $t \in T_l$ **do**
                if t *can be resolved through a precedence constraint* **then**
                    add the precedence constraint that resolves t to $\mathcal{P}_l$
                else
                    break
            if *All threats were resolvable* **then**
                $\pi_l \leftarrow$ copy of π'
                add $\tau, l, \mathcal{P}_l$ to π_l
                add π_l to $\mathcal{P}_a$;
    $P \leftarrow P_\tau$

POP still adds the primitive tasks as part of the expansion in order to maintain the ability to solve problems that cannot be solved solely with the highest level abstract task and maintain completeness.

An abstract task can have multiple methods. The abstract task can be refined into each of these methods. Adding a specific method to a partial-order plan is shown in Algorithm 4. Each method m contains a partial-order plan π_m containing tasks (both abstract and primitive), causal links, and precedence constraints. When a method m is added to

a partial-order plan π, the causal links and precedence constraints from π_m are added to π. The list of tasks from π_m are topologically sorted based on the causal links and precedence constraints. Each task is added to π in a similar manner to Algorithm 2. The main difference is a task originally from π_m may have preconditions that are supported by the causal links from π_m and so new causal links to support these preconditions are not created. Adding these tasks may still cause threats upon the causal links that were originally from π and so new precedence constraints may need to be added. Each of the partial-order plans generated from this process is added as a child node to π. The path cost and heuristic for these nodes are calculated in the same way as for FCPOP using the STN and modified TRPG.

2.6 Experimental Evaluations

2.6.1 Forward Chaining Partial-Order Planner

In order to evaluate the performance of FCPOP, we use eight of the nine benchmarks from the 2018 International Planning Competition's (IPC) temporal track [20] and compare the results of FCPOP against state-of-the-art temporal planners in OPTIC [16] and TFLAP [15, 39], the version of FLAP used for 2018 IPC's temporal track. The *road traffic accident* domain was removed as neither OPTIC nor TFLAP completed any of the problems for it in the competitions. Each domain has 10 problems, so there are 80 benchmark problems in all. As with the 2018 IPC competition, each planner was given 30 minutes and 8 GB of available memory to utilize for each experiment.

Table 2.1 shows how many problems each of the three planners (FCPOP, TFLAP, and OPTIC) solved for each domain as well as the total number of problems that each planner solved. Figure 2.3 shows how the plans created by TFLAP and OPTIC compared to the plans created by FCPOP in terms of makespan. In order to make the data easier to understand, the difference in the makespan of the plan created by either OPTIC or TFLAP and the makespan of the plan created by FCPOP is displayed. As such, $y = 0$ corresponds

Table 2.1: Number of problems solved for each domains. Red denotes the best value for a row.

Domain	FCPOP	TFLAP	OPTIC
Airport	9	9	3
Cushing	7	3	10
Floortile	6	3	0
Mapanalyser	9	8	0
Parking	10	10	8
Quantum Circuit	9	8	8
Sokoban	5	4	3
Trucks	10	10	10
Total (80)	65	55	42
Coverage	79.2%	68.75%	52.5%

to the makespan of plans created by FCPOP and points above $y = 0$ are plans created by TFLAP or OPTIC with longer makespans and below are plans they created with shorter makespans. Figure 2.4 shows the planning speed of TFLAP and OPTIC compared to the planning speed of FCPOP. The difference in the time taken to plan by TFLAP or OPTIC and the time taken to plan by FCPOP is shown. $y = 0$ corresponds to the planning time of FCPOP and points above $y = 0$ are problems in which TFLAP or OPTIC were slower and below are problems in which they were faster. For both figures, only problems that FCPOP and at least one other planner solved are shown.

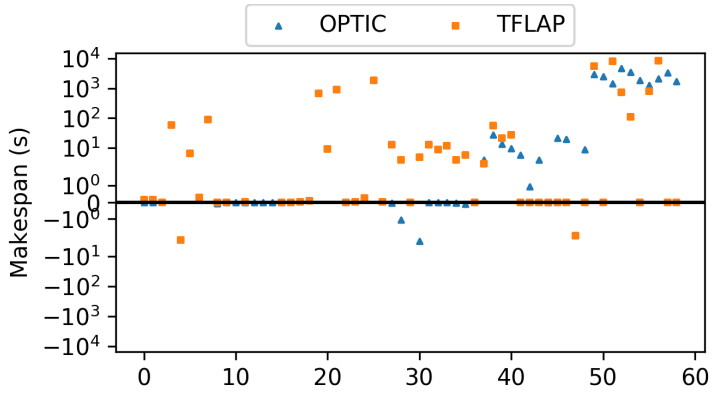


Figure 2.3: $y = 0$ corresponds to the makespan of plans created by FCPOP and points above $y = 0$ are plans created by TFLAP or OPTIC with longer makespans and below are plans they created with shorter makespans.

As can be observed in Table 2.1, OPTIC solved more problems in the *cushing* domain

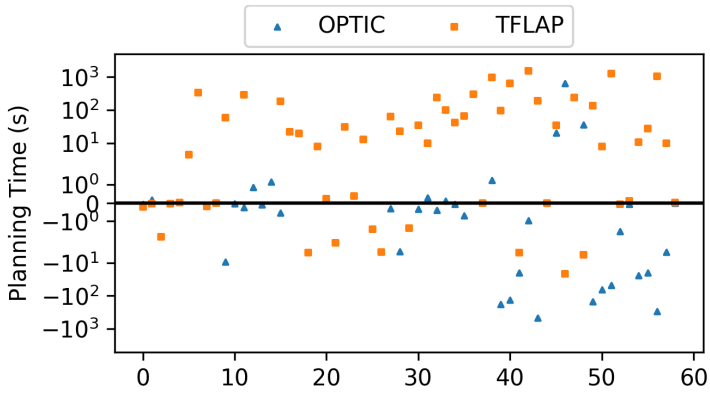


Figure 2.4: $y = 0$ corresponds to the planning time of FCPOP and points above $y = 0$ are problems in which TFLAP or OPTIC were slower and below are problems in which they were faster.

than FCPOP; however, FCPOP solved the same number or more problems in all other domains and solved more problems overall. Also on problems that both FCPOP and OPTIC solved, FCPOP created plans with a better makespan on 66.5% of them and made plans that on average had 83.6% of the makespan of the ones that OPTIC created. These results are likely due in part to FCPOP using a non-relaxed delayed task ordering which makes FCPOP more flexible in how it constructs plans. As it can add a task anywhere in the plan, as opposed to only after the frontier state, this allows for less backtracking and plans with shorter makespans.

OPTIC uses Enhanced Hill Climbing (EHC) as its primary search algorithm, which lets it create a fast, but often lower quality first plan [15]. If EHC cannot find a solution then it switches to a best first search, which unlike FCPOP's A* search, does not take the cost of a node into account when deciding if it should be expanded. This results in nodes that are heuristically close to the goal to be expanded even if the cost of the corresponding partial-order plan is large.

While FCPOP created better plans more than half of the time, OPTIC was faster for 69.2% of the problems that both planners solved. This is likely due to extra computation time being needed by FCPOP in expanding a node. First, OPTIC can check whether the precondition of a task holds in the frontier much quicker than FCPOP can determine if the

task can be placed anywhere in the plan. Second, OPTIC does not need to check for and resolve threats when adding a new task, as the new task is placed after any other tasks that it could have conflicts with. Third, the number of tasks that can be supported by a frontier state is fewer than the number of tasks that be supported throughout a partial-order plan and so OPTIC's branching facter is smaller. These all lead to FCPOP taking about 2.5x the amount of time OPTIC took to plan on average.

In all domains, FCPOP solved the same number or more problems than TFLAP resulting in FCPOP solving more problem overall. Also, FCPOP created a plan for 94.5% of the problems that TFLAP solved. While TFLAP similarly does not relax the delay task ordering, it does not use a temporal heuristic and uses a different algorithm for scheduling. TFLAP uses heuristics and path costs that are based on the number of tasks in the plan. This can lead to TFLAP creating a plan with fewer tasks, but a longer makespan. Also, due to a lack of temporal information in its search guidance, domains with dead-ends cause TFLAP to encounter plateaus and get stuck in areas of the plan space. FCPOP was also faster than TFLAP on 70.9% of the problems that both solved. FCPOP took on average 86.4% of the time that TFLAP took to solve a problem. This also is likely due to FCPOP having a temporal heuristic that allows it a more guided search and lets it search less of the space before finding a solution.

2.6.2 Forward Chaining Hierarchical Partial-Order Planner

FCHPOP was tested on the *satellite* domain first introduced in the 2002 IPC [28], the *survivors* domain presented in [19], and the *turn and open* domain which was in the 2011 IPC [23]. For each of the three domains a random generator was used to create a non-hierarchical version and a hierarchical version of the domain and problem.

The *satellite* domain involves gathering data from a number of scientific imaging instruments on a number of satellites and sending the data to earth. The random generator varies the number of satellites, instruments, stars, planets, interesting phenomena and imaging

modes. The hierarchical tasks allow for domain knowledge to be given to the planner and for it to use premade procedures for sets of tasks that would be repeated.

The *survivors* domain involves sending teams of robots to rescue survivors of a natural disaster and bring them to hospitals. In this domain, the hierarchical tasks allow for planning for the teams as opposed to planning for the individual robots and for zones instead of individual locations. The random generator varies the numbers of teams, hospitals, and survivors as well as the positions of the hospitals and the survivors.

The *turn and open* domain has a number of robots with two gripper hands, multiple rooms with balls that need to be transported, and doors that must be opened between the rooms. The random generator varies the number of robots, balls, and rooms, which adjacent rooms have doors connecting them, and whether doors are open or closed in the initial state.

For each domain, 20 problems were randomly generated. For each problem FCHPOP planned for the hierarchical version of the problem and FCPOP planned for the problem with just the primitive tasks. These experiments were all given 30 minutes to plan and 8 GB of memory.

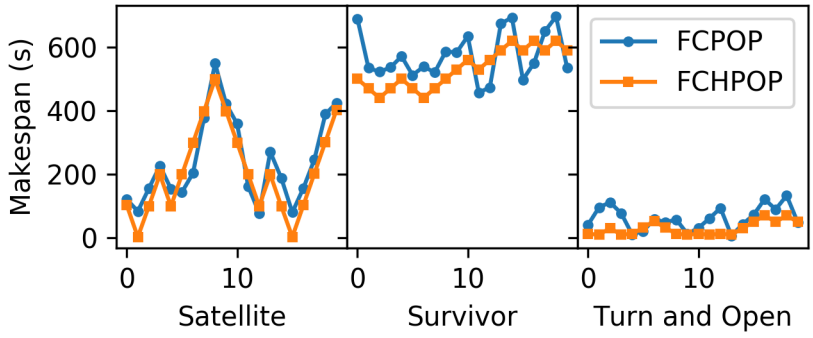


Figure 2.5: Comparison of FCPOP and FCHPOP in terms of plan makespan

FCHPOP on average took 28.1% of the time that FCPOP did for the same problem. FCHPOP also explored 20.8% of the number of nodes and visited 9.2% of the number of nodes that FCPOP did. Using the abstract tasks significantly reduces the number of nodes the FCHPOP needs to visit and explore to solve the problem, which in turn reduces the amount of time need to solve the problem.

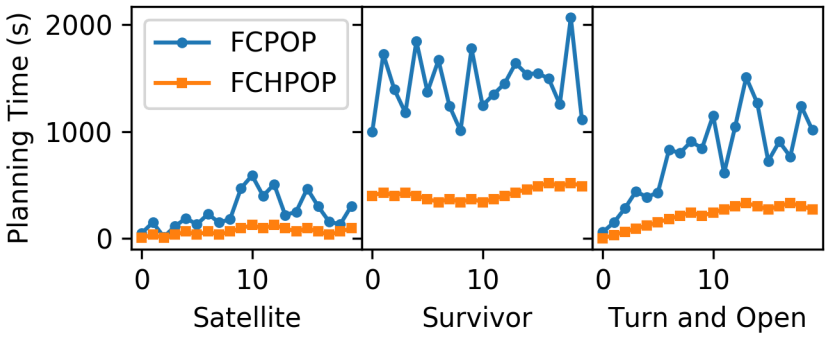


Figure 2.6: Comparision of FCPOP and FCHPOP in terms of planning time

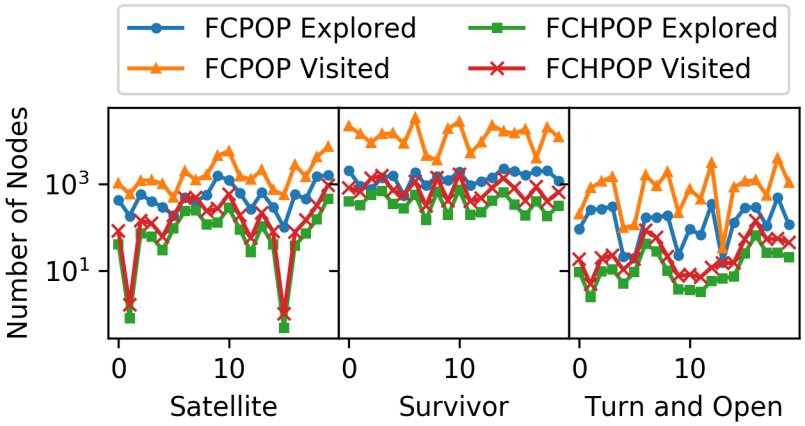


Figure 2.7: Comparison of FCPOP and FCHPOP in terms of number of nodes visited and explored

One of the reasons for this is that abstract tasks provide domain specific information about how a procedure could be done. In a two level hierarchy such as in the *satellite* domain, adding and refining an abstract task involves two node expansions, whereas creating the same effect on the plan without abstract task would require an expansion per task to be added. This effect compounds if an abstract task is added multiple times. In the satellite domain for instance many of the plans repeatedly turn a satellite towards a target, calibrate an image instrument, collect an image, point towards Earth, and then send the data to Earth. For each time this is done, FCPOP has to expand four nodes whereas FCHPOP only needs to expand two. With different combinations of hierarchical levels and number of tasks in a method, the ratio of nodes FCHPOP to nodes FCPOP need to expand to solve the same subproblem changes, but generally FCHPOP needs fewer expansions. This is an advantage

of providing expert domain knowledge. The drawback to this is that if there is a better way to solve a subproblem than the methods in FCHPOP's abstract task then FCHPOP may ignore the better way to solve it and use the precomputed methods of an abstract task. This led to FCPOP creating plans with better makespans in some of the problems.

Another reason that FCHPOP was generally faster than FCPOP in the *survivors* domain was using the hierarchy of the abstract tasks to plan for teams of robots instead of individual robots and for zones instead of individual locations. By creating the hierarchy in this way teams of robots could share a common goal and move together without needed to compute each individual robot's tasks separately. This combined with the previously mentioned procedure led to significant speed up in the speed of planning as well as reducing the number of nodes visited and explored.

2.7 Discussion & Conclusion

We presented two new planning algorithms in FCPOP and FCHPOP, which generate time flexible plans with concurrent tasks. FCPOP combines grounded forward chaining with partial-order planning, fully utilizes the delayed task ordering component of the least-commitment philosophy, and uses a temporal state-based heuristic in a modified version of the Temporal Relaxed Planning Graph. This results in our planner being able to create plans with shorter makespans while still being fast. FCPOP is shown empirically to make plans with equal or shorter makespan than state-of-the-art temporal planners most of the time. FCHPOP builds upon FCPOP by adding hierarchical information in the form of abstract tasks that can be recursively refined. This is shown to speed up the planning process and reduce the number of nodes that need to be visited and explored, all while still generating high quality plans.

However, FCPOP and FCHPOP do have their limitations and there are open questions left to explore. First, while multi-agent planning problems can be cast as temporal planning problems, which allows both planners to solve multi-agent planning problems, FCPOP

and FCHPOP cannot handle allocating arbitrary teams of robots to tasks. This means that FCPOP and FCHPOP create a grounded task for each combination of robots that can participate in a task and as a result FCPOP and FCHPOP do not scale well for coalition-based multi-agent planning. Second, while FCPOP and FCHPOP can utilize finite-domained state variables, they cannot handle numerical state variables, which are fairly prevalent in robotics domains. OPTIC handles linear continuous equations and some simple numerical operations on state variables by utilizing a numerical solver as part of their scheduling algorithm, so a similar approach can be utilized to gain this capability. Third, FCPOP and FCHPOP assume a fully-observable closed-world deterministic environment. By producing a partial-order plan that can be easily converted to an STN, there is some flexibility in the solution for the starting and completion time points. However, real world robotics domains tend to have uncertainty caused by numerous sources.

CHAPTER 3
INTERLEAVING TASK ALLOCATION, SCHEDULING, AND MOTION PLANNING

3.1 Introduction

Whereas in the previous chapter, we focused on interleaving task planning and scheduling for temporal planning, we now transition to another subset of our holistic multi-robot coordination problem. In this chapter, we focus on interleaving task allocation, scheduling, and motion planning for a problem known as trait-based time-extended task allocation[1]. As part of this focus, we introduce a novel framework consisting of three interconnected modules - task allocation, scheduling, and motion planning. A key attribute of our framework is effective information exchange among the different modules through shared constraints (see Figure 3.1). In contrast to solving these problems in sequence, our interleaved approach enables efficient generation of allocations, schedules, and motion plans that do not conflict with each other. We demonstrate that our interleaved approach improves allocation quality, scheduling efficiency, and overall computational efficiency.

In addition, we make specific contributions to task allocation for heterogeneous multi-robot teams. Specifically, we introduce a search-based approach to task allocation, named Incremental Task Allocation Graph Search (ITAGS). ITAGS computes an allocation of robots to tasks while *simultaneously* optimizing the satisfaction of task requirements and the makespan (i.e. execution time) of the associated schedule.

We leverage recent advances in instantaneous task allocation [40, 41], and model our

[1]The material in this chapter is based on:
G. Neville*, **A. Messing***, H. Ravichandar, S. Hutchinson, and S. Chernova, "An Interleaved Approach to Trait-based Task Allocation and Scheduling," International Conference on Intelligent Robots and Systems, 2021.

*Co-First Authors

task requirements in terms of the traits (i.e. capabilities) required for each task. Such trait-based specifications do not require the user to explicitly specify the relationships between each task and robot, allowing for generalization to different types of robots and to different types of tasks.

We enable effective interleaving of task allocation, scheduling, and motion planning by introducing a convex combination of two heuristics in our iterative search algorithm. The first heuristic measures how well the current allocation satisfies the specified trait requirements. The second heuristic measures the efficiency of the schedule associated with the current allocation. We note that our second heuristic accounts for both the feasibility and duration of all motion plans when evaluating a schedule.

Combined, our two heuristics allow ITAGS to prune branches of the incremental task allocation graph that are infeasible to execute due to constraint violations either in scheduling or motion planning. Such pruning allows ITAGS to prefer satisficing incremental allocations, resulting in significant improvements in overall computational efficiency and solution quality. Further, we provide practical insights into the relative influence of each of the two heuristics on ITAG's performance.

To illustrate the impact of our contributions to the robotics community, we evaluate our framework using experiments in a simulated emergency response domain. Firstly, our experiments systematically investigate both individual and combined effects of the two heuristics in terms of convergence, computational cost, makespan, and search efficiency. Secondly, we compare the proposed framework against a baseline approach that does not interleave allocation, scheduling, and motion planning. Thirdly, we compare the proposed framework against two recent state-of-the-art task allocation algorithms. The results of these evaluations conclusively demonstrate the efficacy and necessity of our framework.

3.2 Related Work

A rich body of work has addressed the multi-robot task allocation (MRTA) problem [42, 43] and the closely-related scheduling problem [44]. Our work addresses a variant of the MRTA problem that involves single-task (ST) robots, multi-robot (MR) tasks, and time-extended allocation (TA). As such, we limit our discussion of related work to methods that address the ST-MR-TA variant of MRTA.

A popular approach to solve the ST-MR-TA problem has been the use of auction-based methods in which robots and tasks are matched using a bidding process based on models of how well each robot can perform each task [45, 46, 47, 48]. However, these methods require that each multi-robot task be decomposed into multiple single-robot tasks or that the user specifies the distribution of robots for each task. Additionally, they either schedule tasks then allocate them or vise versa. ITAGS, on the other hand, does not require decomposition of tasks nor a specification on the number of robots needed for each task, and it interleaves task allocation and scheduling, which allows it to create more efficient schedules than possible when operating sequentially.

Optimization methods, while primarily used for solving scheduling and ST-SR-TA variant of the MRTA problem (see [42] for more information on the ST-SR-TA problem), have also been utilized to solve the ST-MR-TA problem [49, 50, 51]. Optimization-based approaches typically cast task allocation as a mixed-integer linear program. However, each of these approaches either does not require all tasks to be completed or requires decomposition of multi-robot tasks. In comparison, ITAGS requires that all tasks be accomplished and does not require the decomposition of tasks.

Recent efforts have formulated the ST-MR-TA problem as tree and graph-search problems [52, 53]. Both of the approaches in [52] and [53] first schedule tasks into a sequence of temporal windows and then allocate robots to these tasks. ITAGS also conducts a tree search, however, does not use temporal windows, which allows for a higher amount of

concurrency between tasks, and simultaneously considers both scheduling and allocation through its heuristics.

Some authors have proposed using processor scheduling techniques to solve the ST-MR-TA problem [54, 55]. Recently, Capezzuto *et al.* [55] proposed Coalition Formation with Improved Look-Ahead (CFLA2) and Cluster-based Coalition Formation (CCF), two algorithms based on processor scheduling techniques, to solve what they call the Coalition Formation with Spatial and Temporal Constraints Problem (CFSTP). These approaches do not require all tasks to be completed when they allocate. On the other hand, ITAGS ensures that all tasks are executed.

While the methods discussed thus far have numerous advantages, a common limitation is that they assume that the desired behavior is specified in terms of an optimal robot distribution or a utility function describing how well each robot can perform each task. In contrast, recent advances have enabled modeling of task requirements in terms of desired trait distributions [56, 40, 41]. These trait-based models are more robust to changes in the number of robots. However, the existing approaches presented in [56, 40] are limited to binary traits. These binary traits are less expressive than continuous traits and lead to less robust models of the agents in a multi-agent team. In addition, the approaches presented in [40, 41] are limited to the ST-MR-IA (instantaneous allocation) problem. They do not consider scheduling, making these solutions unable to handle the time-extended domain of our problems. We take inspiration from these methods and propose an approach for continuous trait-based ST-MR-TA that simultaneously solves task allocation, scheduling, and motion planning.

3.3 Problem Description

Assigning tasks to the different robots and coalitions requires reasoning about their complementary traits and the team's limited resources. These assignments must also respect robot motion planning and scheduling. In this section, we discuss the formulation of a trait-based

time-extended task allocation problem in two parts. First, we present the elements of the problem domain. Second, we describe the format of a solution in this domain.

3.3.1 Problem Domain

Consider a heterogeneous team of K robots, where each robot is defined by its abilities or *traits*, which are modeled as continuous variables. Each robot's traits are defined as

$$\mathbf{q}^{(i)} = \left[\mathbf{q}_1^{(i)},\ \mathbf{q}_2^{(i)},\ \cdots, \mathbf{q}_U^{(i)}\right]$$

where $\mathbf{q}_u^{(i)} \in \mathbb{R}_+$ corresponds to the u^{th} trait for the i^{th} robot. If the i^{th} robot does not have the u^{th} trait (e.g. firetrucks have a water capacity, but other robots may not) then $\mathbf{q}_u^{(i)} = 0$, otherwise it is a positive value. The traits of the entire team are defined by the **robot trait matrix**

$$\mathbf{Q} = \left[\mathbf{q}^{(1)\intercal},\ \cdots,\ \mathbf{q}^{(K)\intercal}\right]^\intercal \in \mathbb{R}_+^{K\times U}$$

with each row corresponding to one robot and each column corresponding to one trait.

A **Task Network** is a directed graph $G = (V, E)$. The vertices V represent a set of tasks. The edges E connect tasks such that an edge $e = \tau_i \rightarrow \tau_j \quad \tau_i, \tau_j \in V$ represents a **precedence constraint** ($\tau_i \prec \tau_j$), or a relationship that ensures that τ_i concludes before τ_j starts [30].

Given the robot trait matrix $\mathbf{Q}$, the team is required to accomplish N tasks from a task network T. Each task in T is defined by the traits required to accomplish it, a static duration, and its initial and terminal configuration (e.g., to move a box, one or more robots need to be at the box's location to start the task and will be at the terminal location of the move upon finishing it). Robots can either complete tasks individually or collaborate on tasks as part of a coalition, depending on their traits. The desired traits for a task are defined as follows:

$$\mathbf{y}^{(i)} = \left[\mathbf{y}_1^{(i)}, \mathbf{y}_2^{(i)}, \cdots, \mathbf{y}_U^{(i)}\right]$$

where $\mathbf{y}_u^{(i)} \in \mathbb{R}_+$ is the u^{th} trait requirement for the i^{th} task. If the u^{th} trait is not required by the i^{th} task then $\mathbf{y}_u^{(i)} = 0$, otherwise it is a positive value. The tasks required by the entire task network is defined by the **desired trait matrix**

$$\mathbf{Y}^* = \left[\mathbf{y}^{(1)\intercal}, \cdots, \mathbf{y}^{(N)\intercal}\right]^\intercal \in \mathbb{R}_+^{N \times U}$$

with each row corresponding to one task and each column corresponding to one trait.

For a robot to participate in completing a task, it needs a collision-free path through its configuration space $\mathcal{C}$ [57] from its current configuration to the initial configuration of the task, and then it needs a collision-free path to the terminal configuration of the task. Each species of robot can have a different free configuration space (e.g. a quadcopter can fly over obstacles that a ground vehicle cannot).

We define the domain for this problem as the tuple $\mathcal{D} = \langle T, \mathbf{Q}, \mathbf{Y}^*, \mathcal{C}_T, I_c, W \rangle$ where

- T is a task network,
- $\mathbf{Q}$ is the robot trait matrix,
- $\mathbf{Y}^*$ is the desired trait matrix,
- $\mathcal{C}_T = \{\langle \mathcal{C}_{\tau_1}^{init}, \mathcal{C}_{\tau_1}^{term} \rangle, \ \ldots, \ \langle \mathcal{C}_{\tau_N}^{init}, \mathcal{C}_{\tau_N}^{term} \rangle\}$ is the set of initial and terminal configuration spaces for each task in T,
- $I_c = \{c_1, \ldots, c_K\}$ is the set of the initial robot configurations, with one for each robot
- W is the world describing the static geometric information about the environment

3.3.2 Solution Specification

Given the above problem domain, we now introduce the various parts of the solution.

An **allocation** $\mathbf{A}$ for K robots and N tasks is a $N \times K$ matrix where

$$\mathbf{A}_m^n = \begin{cases} 1 & \text{if the } n^{th} \text{ robot is assigned to the } k^{th} \text{ task.} \\ 0 & \textit{otherwise.} \end{cases}$$

Given the problem domain $\mathcal{D}$, we wish to compute a **solution** $\mathcal{S} = \left\langle \hat{\mathbf{A}},\ \hat{X},\ \hat{\sigma} \right\rangle$ where

- $\hat{\mathbf{A}}$ is an allocation that satisfies the desired traits of T
- $\hat{X}$ is a set of discrete trajectories, one for each robot; as part of these trajectories, each robot moves to the tasks it was allocated in the order they were scheduled while avoiding collisions with other robots and the environment; the other trajectories in the set represent the movements needed to execute the tasks in T
- $\hat{\sigma}$ is a schedule with the minimum makespan C. This schedule includes the time to execute the motion plans in X and is temporally consistent

3.4 Communication Between Sub-Problems

Our approach involves solving multiple sub-problems and to reduce computation, we exploit the fact that as each sub-problem is solved constraints are generated on and between the same shared components (i.e., tasks and robots) and these constraints can be shared between the layers working on each sub-problem. This approach utilizes both *positive constraints*, which are constraints based on what *can* be satisfied, and *negative constraints*, which are constraints based on what *cannot* be satisfied. Methods that create positive constraints usually have a a database of constraints paired with their satisfying assignment. This is sometimes known as memoization. Alternatively, methods that create negative constraints usually focus on identifying counterexamples.

Additionally, for our approach we expand constraints to their most general context. For example, if we determine that a specific robot cannot contribute to the execution of a task because its traits cannot contribute, it is possible that that constraint of "robot r cannot contribute to task τ_y" can be generalized to "species s cannot contribute to task τ_y" or if we find a motion plan between two configurations for a specific robot it can be utilized by other robots of the same specifies. Furthermore, if we have 3 species and independently we determine that all 3 species cannot contribute to a task, we can conclude that the task cannot be accomplished at all and so the problem is unsolvable. More constraints and

generalizations will be discussed in the next section.

3.5 Incremental Task Allocation Graph Search

This section outlines the high-level algorithmic architecture that we use for trait-based time-extended task allocation for groups of heterogeneous robots. After introducing the high-level architecture, the following subsections will explain individual layers of the hierarchy.

It is assumed that the task network (T), desired trait matrix ($\mathbf{Y}^*$), the robot trait matrix ($\mathbf{Q}$), a set of initial and terminal configurations spaces for the tasks in T ($\mathcal{C}_T$), a set of the initial robot configurations (I_c), and the world describing the static geometric information about the environment (W) are provided as presented in Section 3.3.

The task allocation layer conducts a heuristic graph search through an incremental task allocation space to find an allocation that satisfies the desired trait requirements of the task network. During the search, it provides allocations to the scheduling layer, which informs the heuristic that guides its search and allows it to prune areas of the incremental task allocation space that are not feasible to schedule. The scheduling layer computes the schedule with the minimum makespan for each allocation. The motion planning layer uses a generic motion planner to determine if there is a feasible motion plan between two configurations for a robot or coalition based on provided configuration spaces as well as the execution time for the motion plan.

This hierarchical method iterates between the layers until it finds an allocation that satisfies the trait requirements of the task network and is feasible to schedule while respecting the execution times for the motion plans (see Figure 3.1). It then outputs the task allocation $\mathbf{A}$, the associated schedule σ, and required motion plans X needed for the schedule. Algorithm 5 contains the pseudo-code for ITAGS.

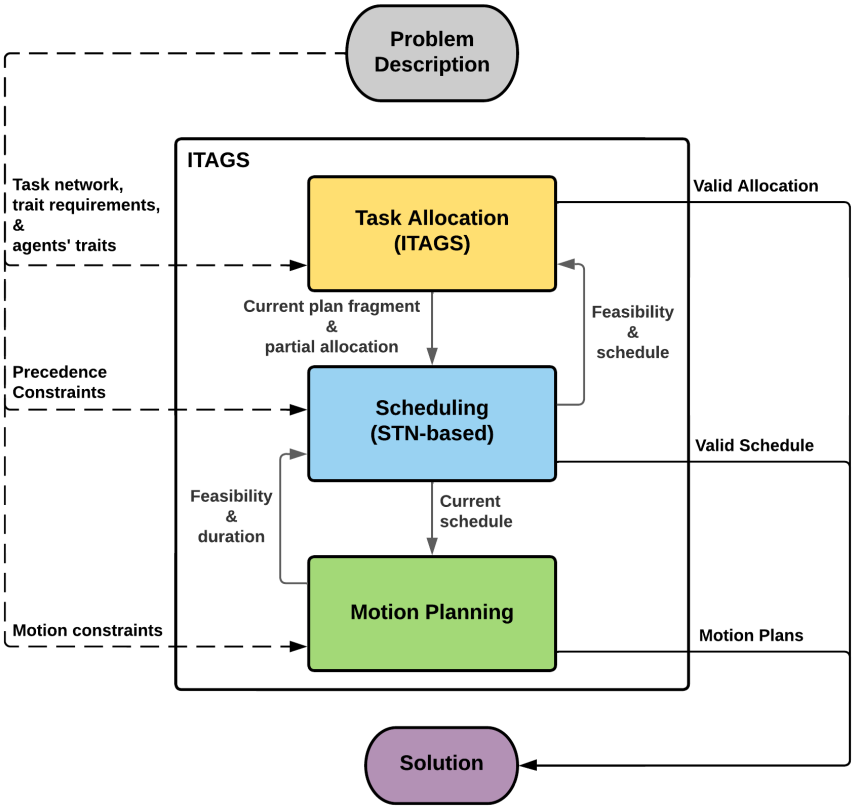


Figure 3.1: High-level architecture of the hierarchical framework.

3.5.1 Task Allocation

The task allocation layer performs a greedy best-first search through the incremental task allocation space. In this space, each node represents an allocation of robots to tasks. Each node is connected to other nodes that differ only by the assignment of a single robot (see Figure 3.2). This graphical representation allows our search to start from an initial node with no allocated robots and to incrementally add robots until an allocation that satisfies the desired trait requirements of the task network is found.

To guide the search, we have developed two heuristics: *Allocation Percentage Remaining*, which guides the search based on the quality of the allocation, and *Normalized Schedule Quality*, which guides the search based on the quality of the makespan of the schedule associated with the allocation. We use a convex combination of the two heuristics, which we call *Time-Extended Task Allocation Quality*.

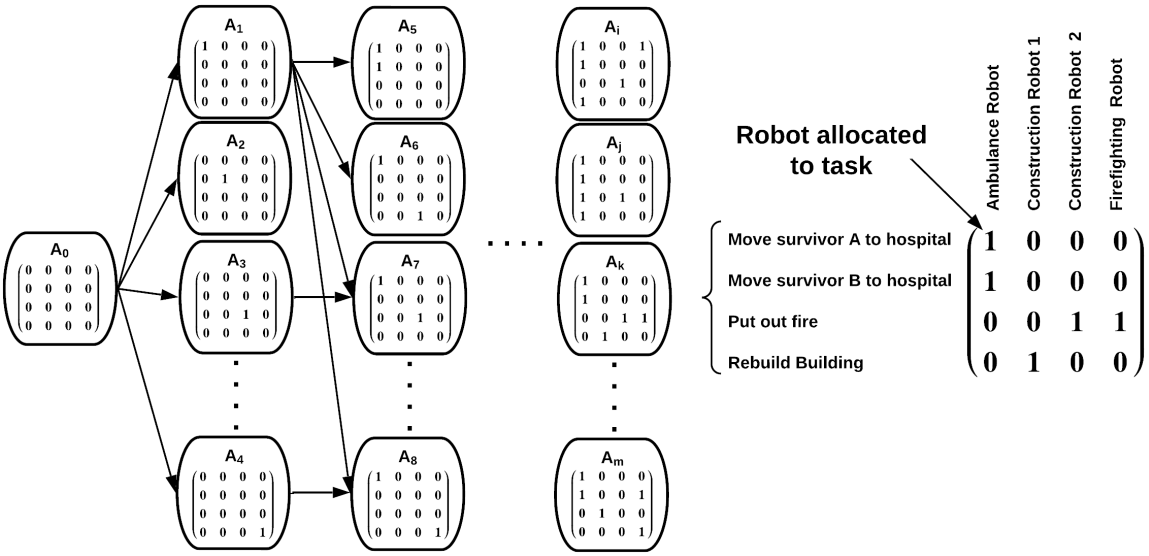


Figure 3.2: An example incremental task allocation graph.

Algorithm 5: ITAGS

Input: T, $\mathbf{Q}$, $\mathbf{Y}^*$, $\mathcal{C}_T$, I_c, W, α
Output: $\hat{\mathbf{A}}$, $\hat{X}$, $\hat{\sigma}$

root $\leftarrow$ empty allocation
pq $\leftarrow$ PriorityQueue({root})
while *pq is not empty* **do**
    // Identify the node with lowest TETAQ value
    node $\leftarrow$ pq.pop()
    // Check if the node is a solution
    if *node.apr == 0* ***and*** *node.nsq* $< \infty$ **then**
        return *node.*$\mathbf{A}$*, node.*X *node.*σ
    // Compute heuristics for each successor
    for *child* $\in$ *generateSuccessors(node)* **do**
        // Compute using Equation 3.1
        child.apr $\leftarrow$ allocationPercentageRemaining(child, $\mathbf{Q}$, $\mathbf{Y}^*$)
        // Compute schedule and motion plan
        σ, σ_{LB}, σ_{UB}, $X \leftarrow$ schedule(child, T)
        // Compute using Equation 3.2
        child.nsq $\leftarrow$ normalizedScheduleQuality(σ, σ_{LB}, σ_{UB})
        // Compute using Equation 3.3
        child.tetaq = $\alpha *$ child.apr $+ (1 - \alpha) *$ child.nsq
        pq.push(child)
return *null*

Allocation Percentage Remaining

Allocation Percentage Remaining (APR) is defined as the percentage trait mismatch error. Specifically, APR is calculated as

$$f_{APR}(\grave{\mathbf{A}}) = \frac{||\max(E(\grave{\mathbf{A}}),\ 0)||_{1,1}}{||\mathbf{Y}^*||_{1,1}} \tag{3.1}$$

where $\grave{\mathbf{A}}$ is the allocation that the heuristic is evaluating, $E(\grave{\mathbf{A}}) = \mathbf{Y}^* - \grave{\mathbf{A}}\ Q$, and $||\cdot||_{1,1}$ is the element-wise l_1 norm. In this equation, $(\grave{A}Q)_{i,j}$ is the summation of the j^{th} trait from each of the robots assigned to the i^{th} task. By taking the difference between the desired trait matrix $\mathbf{Y}^*$ and $\grave{A}Q$, we get the *trait mismatch matrix* E. When $E_{i,j} < 0$ then the coalition assigned to the i^{th} task exceeds the required trait value for the j^{th} trait. An element-wise max operation is performed between E and the zero matrix. This removes all the values in E which represent that an allocated coalition exceeds a required trait value and leaves the values that represent that a required trait value has not yet been met. An element-wise summation is then performed on the resulting matrix to compute the *trait mismatch error*. This error is then normalized by the element-wise summation of the desired trait matrix $\mathbf{Y}^*$ to compute the percentage trait mismatch error. When the error is zero, then the allocation satisfies the desired traits matrix.

Before search, we can quickly calculate the APR when all robots are assigned to all tasks. If the value is not zero, then the problem has no solution. Furthermore, upon the first expansion from the root node, we can quickly determine which species of robots cannot contribute to each task. These constraints decrease the branching factor for the remainder of the search, increasing efficiency.

Normalized Schedule Quality

Normalized Schedule Quality (NSQ) is a measure of how much an allocation minimizes the makespan of its accompanying schedule. Specifically, NSQ is calculated as

$$f_{NSQ}(C_{\grave{\sigma}}) = \frac{C_{\grave{\sigma}} - C_{\sigma_{LB}}}{C_{\sigma_{UB}} - C_{\sigma_{LB}}} \tag{3.2}$$

where C_σ is the makespan, or completion time, of the schedule σ, $\grave{\sigma}$ is the schedule based on T and $\grave{\mathbf{A}}$, σ_{LB} is the schedule without any constraints placed on the schedule from the allocation and motion planning, and σ_{UB} is the schedule where the task network is totally-ordered and all motion plans are assumed to be the longest possible length.

The three variables $\grave{\sigma}$, σ_{LB}, and σ_{UB} are all computed by the scheduling layer. As NSQ only considers the schedule and not the allocation, it tends to favor a broader search. This is caused by nodes closer to the root having fewer constraints and, therefore, lower makespans. This leads to NSQ finding an allocation that satisfies the desired trait requirements with the minimum makespan at the expense of searching a much larger area of the graph.

Time-Extended Task Allocation Quality

Using a convex combination of APR and NSQ, we create a heuristic that considers both the quality of the allocation as well as the quality of the schedule generated from it. Time-Extended Task Allocation Quality (TETAQ) is calculated as

$$f_{TETAQ}(\grave{\mathbf{A}}, C_{\grave{\sigma}}) = \alpha f_{NSQ}(C_{\grave{\sigma}}) + (1-\alpha) f_{APR}(\grave{\mathbf{A}}) \tag{3.3}$$

where $\alpha \in [0, 1]$ is a user-specified parameter that controls each heuristic's relative influence. If $\alpha = 0$ then this heuristic is APR, and if $\alpha = 1$ then the heuristic is NSQ. TETAQ takes qualities from both to perform a search, which allows it to balance finding an alloca-

tion that satisfies the desired traits quickly with finding one that minimizes the makespan of the assigned robots' schedule.

3.5.2 Scheduling and Motion Planning

The scheduling layer determines if it is feasible to find a schedule for a task network and an allocation and provides the makespan used by the heuristics. This layer builds the schedule based on three temporal components: the static duration of each task, the time needed for each robot to transition between two successive tasks, and the time needed for the assigned coalition of robots to execute the movements required to complete the task. To this end, the scheduling layer provides pairs of configurations to the motion planning layer.

The motion planning layer uses the information received from the scheduling layer to determine if there is a feasible motion plan between the two configurations. If it can construct a motion plan, then the motion plan is sent back to the scheduling layer. To reduce computational costs and reuse motion plans, this layer memoizes the path for each specific pair of configurations and specific robot or coalition.

As mentioned in the previous section, the memoization creates a positive constraint containing both the query with the two configurations and the motion plan found. The motion plan can be utilized by other robots of the same species. Additionally, the motion planning layer can generate negative constraints if it cannot construct a motion plan (if infeasibility can be determined by the motion planning algorithm chosen) or if the motion planning layer times out. If the motion planning query that failed was for the initial transition between a robot r_i's initial configuration and its first task τ_j, then this solely creates the constraint that robot r_i cannot participate in task τ_j. During the first expansion of the root node, we can quickly check which specific robots can reach the initial configuration for a task and determine which robots cannot participate in each task. This reduces the branching factor for the remainder of the search and increases efficiency. If the motion planning query was for part of the execution of a task τ_j, then a constraints can created that

states that robot r_i's species cannot participate in task τ_j. If the motion planning query was for a transition between tasks τ_j and τ_k, then the constraint created states that robot r_i's species cannot participate in both tasks τ_j and τ_k, however the are no constraints place on the robot r_i's species participating in each task individually. This is because its possible the robot's ability to reach either of τ_j or τ_k depends on its previous location (e.g., the robot could be good at pushing doors open, but struggle to pull doors open making motion plans irreversible).

If a motion plan is successfully calculated, the scheduling layer then uses the coalition's speed to determine the time needed to execute the motion plan. If a feasible schedule can be found then it calculates $\grave{\sigma}$, σ_{LB}, and σ_{UB}.

For a time-extended task allocation, a schedule has two different types of temporal constraints: precedence constraints and mutex constraints. The precedence constraints come from the task network. A **mutex constraint** is a temporal relationship between two tasks, τ_i and τ_j, that ensures that either τ_i must finish before τ_j starts or τ_j must finish before τ_i starts. A mutex constraint is created when a robot is assigned to a task τ_i making it unable to perform another task τ_j at the same time as τ_i.

To find a schedule that adheres to both of these types of constraints, the scheduling layer uses a three-part scheduling approach. The first part converts the task network into a Simple Temporal Network (STN) [33], which provides a graphical representation of the start and end times for each task, where tasks are separated by precedence constraints. STNs are commonly used for scheduling due to their ability to be updated and checked for consistency in polynomial time [58]. We use a variant of the single-source shortest path algorithm [33] to compute the minimum makespan schedule from this STN. This schedule does not yet include the durations of any motion plans nor considers the mutex constraints imposed by the allocation, and so this schedule is σ_{LB} as used by NSQ.

The second part of this approach adds in the mutex constraints from the allocation. In order to create a schedule, the mutex constraints need to be converted into precedence

constraints by selecting one of their two orderings (i.e. for a mutex constraint between two tasks τ_i and τ_j selecting that either τ_i must finish before τ_j starts ($\tau_i \prec \tau_j$) or vise versa). We perform a tabu search [59] over the possible orderings for each mutex constraint while minimizing the makespan of the schedule. Each node in this search is the STN for σ_{LB} with additional edges added for the precedence constraints created from the ordering selection of the mutex constraints. The resulting STN from this part of the approach is called STN$_d$.

The third part of this approach adds in the execution durations of the motion plans to the STN. During this step, the motion planning layer determines if every required motion plan is feasible. If feasible, the length of each motion plan is used by the scheduling layer to determine the execution duration of the motion plan.

If it is not feasible to find a schedule for an allocation, then the task allocation layer is alerted, and the allocation is pruned from the search. Finding a schedule is infeasible if:

- the STN for σ_{LB} is temporally inconsistent
- no temporally consistent STN can be found during the tabu search
- the motion planner cannot find a solution (if infeasibility can be determined)
- the motion planner times times out

Constraints generated by the motion planning layer were discussed earlier, however, if the STN for σ_{LB} is temporally inconsistent then a solution cannot be found as the original task network is not a directed acyclic graph and if no temporally consistent STN can be found during the tabu search then the allocation node is pruned from the tree.

Finally, we compute the makespan of σ_{UB}. For computational efficiency, we approximate an over-estimation of the makespan of the worst possible schedule without having any robots slow down or wait,

$$C_{\sigma_{UB}} = \frac{2Mz}{w} + \sum_{m=1}^{M} dur(\tau_m)$$

where N is the number of tasks in T, $dur(\tau_m)$ is the static duration of task τ_m, z is the

length of the longest possible path in W, and w is the speed of the slowest robot.

3.6 Experimental Evaluations

We evaluated ITAGS using three sets of experiments in a simulated emergency response domain [60, 19, 61, 62, 63]. In this domain, a diverse set of robots with different traits need to work together to rescue wounded survivors, deliver medicine to hospitals, put out fires, and rebuild damaged infrastructure.

For the first two experiments, we generated a set of 105 problems from this domain by randomly sampling the number of robots, survivors, fires, and damaged buildings. Each problem had between 6-12 robots and 12-45 tasks. We also randomized the locations of the survivors, fires, damaged buildings, hospitals, and the robots' initial location. In the first set of experiments, we evaluated the effectiveness and the relative influence of our heuristics. In the second set of experiments, we evaluated the effectiveness of ITAGS's interleaved allocation and scheduling by comparing it to a baseline that uses a sequential version of ITAGS. For both experiments, we report computation time, the number of nodes expanded and visited, and the makespan of the generated schedule as metrics.

For the third experiment, we generated a set of 50 problems from this domain. Each problem had 20 robots and 40 tasks. The problems were randomized similarly to the problems for the first two experiments. For this experiment, we compared ITAGS against two state-of-the-art ST-MR-TA task allocation algorithms: CFLA2 and CCF [55]. These algorithms represent the most recent efforts in solving the ST-MR-TA task allocation problem.

For all of the experiments, maps from the Robocup Rescue Competition [60] were used. For the motion planning layer, we used a Lazy PRM [64] implementation from the Open Motion Planning Library [65]. We ran all experiments on an i7-8565 CPU with 16GB of RAM.

3.6.1 Relative Influence of APR and NSQ on Performance

The first experiment involved ablation studies to investigate the relative influence of APR and NSQ on performance. To this end, we generated five variants of ITAGS, each with different values for α ($0,\ 0.25,\ 0.5,\ 0.75$, and 1) from Equation 3.3. Note that these values for α indicate different relative weightings of NSQ and APR. The results of these tests can be seen in Figure 3.3.

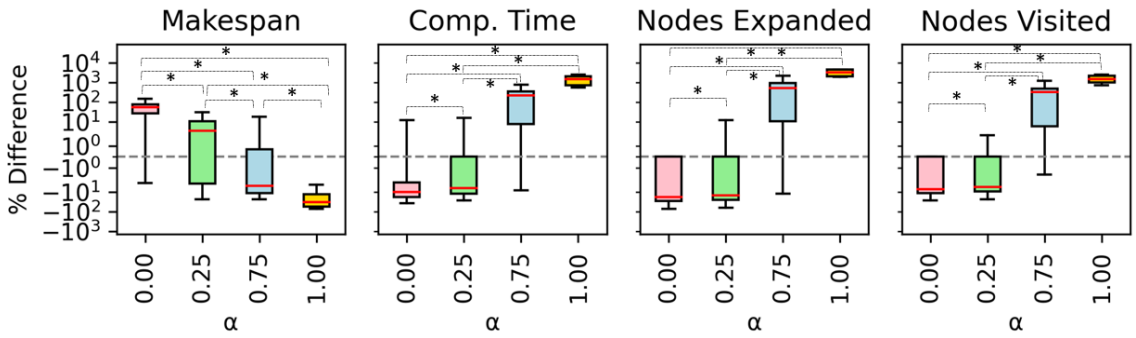


Figure 3.3: The results of ITAGS with various α values normalized with respect to ITAGS$_{\alpha=0.5}$. $y = 0$ corresponds to ITAGS$_{\alpha=0.5}$. Anything above $y = 0$ is worse than ITAGS$_{\alpha=0.5}$ and conversely anything below is better. '*' denotes statistical significance with a p-value < 0.05.

To make the relationship between the weighting of the heuristics and the performance clear, Figure 3.3 displays the results normalized to an equal weighting of both heuristics ($\alpha = 0.5$). As such, the horizontal line for $y = 0$ represents the results of $\alpha = 0.5$. Anything above $y = 0$ represents a result that was larger than the $\alpha = 0.5$ baseline (e.g. took more time to compute or expanded more nodes), and anything below $y = 0$ represents a result that was smaller than the $\alpha = 0.5$ baseline (e.g. took less time to compute or expanded fewer nodes).

All α values were capable of solving all 105 problems except $\alpha = 1.0$, which only solved 7.5% of the problems. As $\alpha \to 1$, it causes the search to focus more on minimizing the makespan, thereby mimicking a breadth-first search. This is because adding more robots increases the number of constraints and, subsequently, the makespan. As a result of these broader searches, ITAGS$_{\alpha=1}$ is more likely to run out of memory and fail to solve the

problem.

We performed a Kruskal-Wallis test followed by a post-hoc Dunn's test to show the statistical significance of each of the tested α values. These tests' results are shown in Figure 3.3 with '*' denoting p-values < 0.5. These results indicate that α changes have a statistically significant effect on the computation time, makespan, the number of nodes expanded, and the number of nodes explored.

We find that as α increased, computation time increased, and makespan decreased. Specifically, as $\alpha \to 1$, the search tends to produce solutions with lower makespans. However, it also tends to require higher computation times as a result of visiting and exploring more nodes. Conversely, as $\alpha \to 0$, the search tends to visit and explore fewer nodes, leading to lower computation times. However, this comes at the expense of a longer makespan. These experiments show that there is a balance between minimizing makespan and minimizing computation time. Our combined heuristic TETAQ merges the effects of the two heuristics to balance both computational efficiency and solution quality.

3.6.2 Effects of Interleaving on Performance

In this experiment, we created a sequential version of ITAGS, known as ITAGS$_S$. Instead of using scheduling and motion planning to guide the search for task allocation, ITAGS$_S$ completes each operation in sequence. It searches the allocation graph until an allocation that satisfies the trait requirements is found. If possible, ITAGS$_S$ schedules the found allocation and updates the schedule with the execution times of motion plans. If either a schedule cannot be created because the allocation is temporally inconsistent or one of the motion planning queries is infeasible, then ITAGS$_S$ continues the search until a satisficing allocation is found.

The results of the comparison can be found in Figure 3.4. Similar to the first experiment, we normalized the metrics with respect to ITAGS$_{\alpha=0.5}$. As can be seen, ITAGS$_{\alpha=0.5}$ consistently outperformed the sequential version ITAGS$_S$ in terms of all of the metrics.

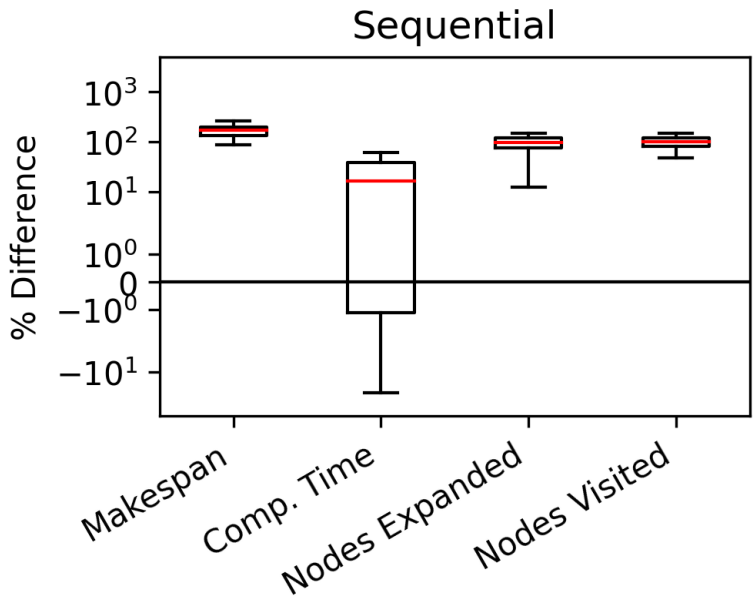


Figure 3.4: The results of the sequential version of ITAGS (ITAGS$_S$) normalized with respect to ITAGS$_{\alpha=0.5}$. $y = 0$ represents ITAGS$_{\alpha=0.5}$. Anything above $y = 0$ is worse than ITAGS$_{\alpha=0.5}$ and conversely anything below is better.

On average, ITAGS$_S$ generated an output with a 168% longer makespan while taking 17% longer to compute a valid result. It also visited 99% more nodes and explored 97% more nodes on average.

There are two notable reasons for ITAG's superior performance. First, ITAGS uses shared constraints to decrease branching factor and to prune portions of the search tree. In contrast, ITAGS$_S$ is likely to end up exploring branches in the allocation graph that violate scheduling or motion planning constraints. Second, ITAGS also minimizes the makespan of the schedule. However, ITAGS$_S$ does not consider the schedule while searching for an allocation. This leads ITAGS$_S$ to prefer allocating most tasks to robots with higher trait values. As such, ITAGS$_S$ results in a reduced number of concurrent tasks.

3.6.3 Comparison against CFLA2 and CCF

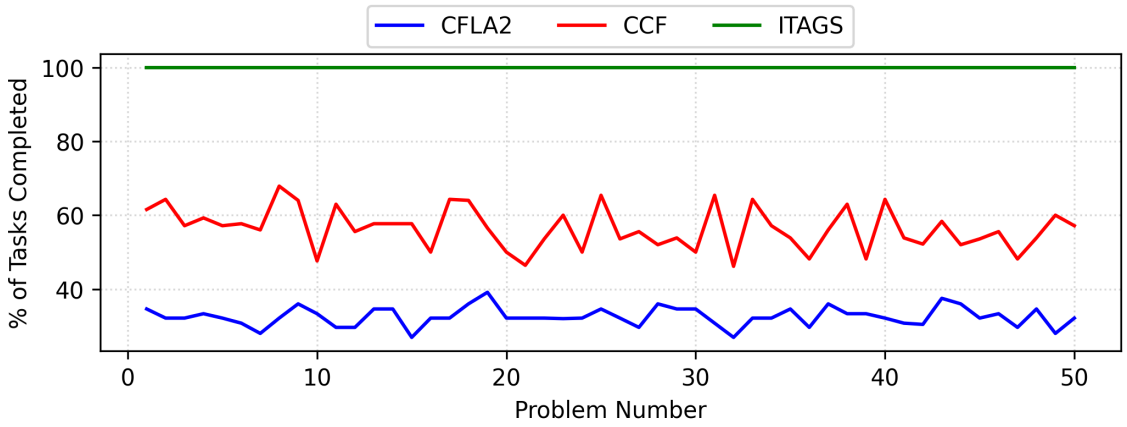
In the third experiment, we compared ITAGS against two state-of-the-art ST-MR-TA task allocation algorithms in CFLA2 and CCF [55]. Both of these algorithms operate on a more-

restricted problem structure that does not involve trait-based agent/task modeling, ordering constraints, or non-graph-based motion planning. Thus in order to benchmark against these algorithms a preprocessing stage was added.

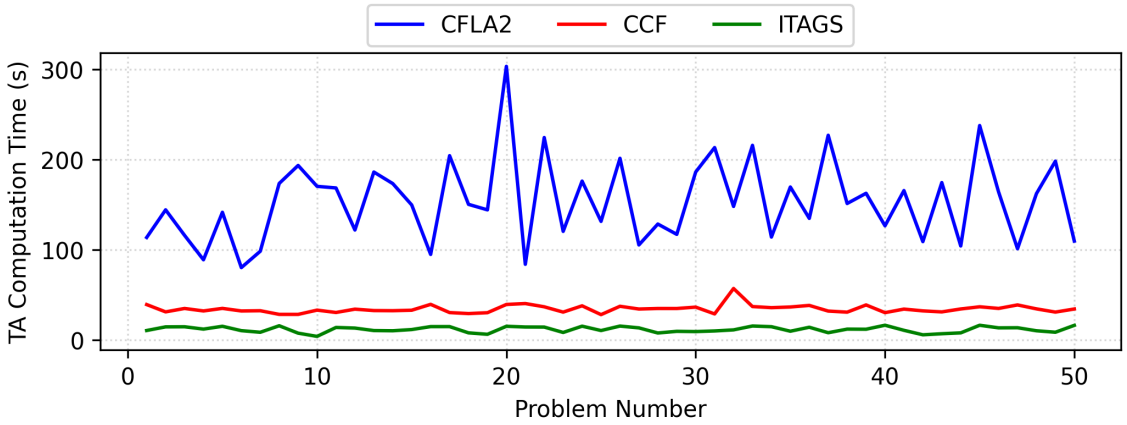
For each problem, the preprocessing stage first generates a fully connected graph where each node is a location (i.e. the hospital, survivor 1's initial location, etc.) and each edge is labeled with the time required to traverse for each robot/coalition. In order to calculate the traversal times, the motion planning module from ITAGS is queried for motion plans between each pair of nodes. The length of each motion plan and the speed of each robot/coalition is then used to compute the traversal times for each edge.

Next, the preprocessing stage accommodates for the fact that the baselines cannot handle trait-based models. The baselines model each task t as having a certain amount of work that needs to be accomplished (w_t). They also have a utility function $u(t, c)$ which computes how fast a specific coalition c can work on a specific task t. The duration of a task t can be computed as $dur(t, c) = w_t/u(t, c)$. The preprocessing stage sets w_t to a constant value for all tasks. It then creates the utility function $u(t, c)$ such that the $dur(t, c)$ is the same as in the problem description for all tasks. If a coalition's collective traits do not satisfy the task's requirements then the utility function returns zero.

As shown in Figure 3.5a, ITAGS is able to allocate and schedule all tasks for each of the problems. This observation is explained by the fact that, unlike CFLA2 and CCF, ITAGS requires allocating and scheduling of all tasks. Indeed, both baselines fail to fully allocate and schedule any of the problems, with CFLA2 and CCF averaging 32.4% and 56.4% of tasks allocated and scheduled, respectively. We observe that the baselines sometimes allocate agents to a task without accounting for the task's trait requirements. This leads to incomplete tasks as robots can get stuck on certain tasks without contributing to the tasks' completion. Additionally, without deadlines, CFLA2's look-ahead phase effectively results in a random choice and impedes its ability to allocate robots to tasks. CCF assumes that every assignment improves task progress irrespective of the current set of robots assigned



(a) The percentage of tasks that each algorithm was able to allocate for each problem.



(b) The amount of time spent by each algorithm on everything except motion planning for each problem.

Figure 3.5: Benchmark against CFLA2 and CCF

to the task. However, this assumption does not hold for tasks in which robots cannot make progress without crossing a minimum threshold in terms of aggregated traits. For instance, consider a task that involves moving a 10 kg object. Assigning a single robot with a 5 kg payload will not contribute to 50% task completion.

Figure 3.5b shows the total computation time excluding the time needed to generate the fully connected graph for CFLA2 and CCF and the time spent motion planning for ITAGS. This ensures that we are comparing only the task allocation and scheduling capabilities of the algorithms. As seen, ITAGS allocates and schedules all tasks while taking less time to compute a solution. On average, ITAGS spends 11.54s on task allocation and scheduling

when computing a solution, while CFLA2 and CCF spend 153.61s and 34.27s, respectively. There are a few reasons that ITAGS performs better than both CFLA2 and CFF. First, ITAGS only considers the start and end time-points for each task when scheduling, whereas both CFLA2 and CCF step through time with discrete timesteps and greedily allocate robots to tasks. Second, ITAGS focuses on simultaneously minimizing the makespan and trait mismatch error. Being able to consider the traits allows ITAGS to model the relationship between agents and task better, leading to more efficient searching of possible allocations. On the other hand, CFLA2 and CCF focus on allocating as many tasks at a time as possible without considering whether the robot's traits contribute to satisfying the trait requirements of a task. CCF specifically also prioritizes allocating a robot to tasks such that travel time is minimized. Third, ITAGS considers the entire schedule as it solves the problem. This allows ITAGS to consider actions throughout the schedule and how they affect the overall makespan. CCF only considers a single timestep at a time, and as a result some of its allocations inadvertently create a bottleneck for future allocations. CFLA2 does have a look-ahead process, but with no deadlines, it effectively results in a random choice. Furthermore, it creates considerable computational burden for this result. The look-ahead process and making allocations for only a single timestep have a detrimental impact on computation time.

3.6.4 Summary

The first experiment empirically demonstrates the trade-offs involved in prioritizing either *Allocation Percentage Remaining* or *Normalized Schedule Quality*. Results from the second experiment indicate that the hierarchical interleaved approach of ITAGS consistently outperforms a baseline approach that sequentially performs allocation, scheduling, and motion planning. In the third experiment, ITAGS is shown to empirically outperform state-of-the-art ST-MR-TA task allocation algorithms CFLA2 and CCF [55].

3.7 Discussion & Conclusion

In this chapter, we introduced a unified framework that interleaves task allocation, scheduling, and motion planning for heterogeneous multi-robot systems. We also presented the iterative search method for solving the trait-based time-extend task allocation problem. To guide the search and enable interleaving, we developed two heuristics, one based on the quality of allocation and another based on the time needed to execute the associated schedule and motion plans. Further, we empirically demonstrated the trade-offs involved in choosing the relative weighting of the two heuristics. Our experiments in a simulated emergency response domain conclusively demonstrate the effectiveness and the relative advantages of our interleaved approach over a sequential baseline and two state-of-the-art approaches.

However, ITAGS does have its limitations and there are open questions left to explore. First, the completeness of ITAGS depends on the completeness of the motion planning algorithm selected to utilized by the motion planning layer. If the motion planning algorithm is complete then ITAGS is complete. However, if a sampling-based motion planning algorithm is used that is probablistically complete, then ITAGS is not complete as the infeasibility of a motion plan cannot be determined. Recent and ongoing research is exploring general approaches for constructing proofs of motion planning infeasibility [66, 67, 68]. These approaches could be incorporated to make ITAGS a complete algorithm regardless of motion planning algorithm. Second, ITAGS assumes that traits are static, however, in real world scenarios traits can change over time or based on external factors such as battery drain. How to integrate and reason about dynamic traits remains an open question for future research. Third, ITAGS assumes deterministic traits and trait requirements, however, in the real world, robots of the same species could have different traits from one another due o different conditions (e.g., wear and tear, battery life, sensor failure, etc). Also, knowing the exact traits required to execute a task is not always viable. How to reason about uncertain

robot traits and task trait requirements for more robust solutions remains an open questions for future research.

CHAPTER 4

INTERLEAVING ALLOCATION, PLANNING, AND SCHEDULING

4.1 Introduction

In this chapter, we move our focus to the full holistic heterogeneous multi-robot coordination problem that we discussed in Chapter 1[1]. This problem interleaves all four subproblems (task planning, task allocation, scheduling, and motion planning) for a problem known as Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) (explained in more detail down below).

We merge elements from Forward Chaining Partial-Order Planner (FCPOP) and Incremental Task Allocation Graph Search (ITAGS) described in the previous two chapters to develop a unified and interleaved framework named Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling (GRSTAPS) as an initial approach to the STAP-STC problem. GRSTAPS tackles the STAP-STC problem by effectively interleaving the execution of its four modules – task planning, task allocation, scheduling, and motion planning. As in previous chapters, an alternative approach would be to combine state-of-the-art solutions to the individual problems such that they operate in sequence. However, such a sequential approach is incomplete and would be forced to backtrack and recompute solutions at all preceding levels when a particular level fails to find a solution. For example, the sequential approach could create a symbolic plan and allocate appropriate robots, only to discover that it is impossible to compute a schedule for the given allocation such that all required constraints are satisfied.

[1]The material in this chapter is based on:
A. Messing[*], G. Neville[*], S. Chernova, S. Hutchinson, and H. Ravichandar, "Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling," The International Journal of Robotics Research, vol. 41, no. 2, pp. 232-256, 2022.
[*]Co-First Authors

To avoid frequent backtracking, our interleaved approach incentivizes compatibility among solutions generated at different levels as well as shares constraints on and between tasks and robots. Employing rigorous quantitative analysis, we demonstrate that our interleaved approach significantly improves the overall framework's computational efficiency compared to its sequential counterpart.

In addition to serving as a unified framework to solve STAP-STC problems, GRSTAPS includes innovations at both the task planning and the task allocation levels. At the task planning level, we present an approach for *agent-agnostic partially-ordered* planning. Our planning approach is agent-agnostic in that it determines the tasks necessary to reach the goal state without committing any agents. As such, our plans only include a set of tasks along with necessary ordering constraints. This reduced commitment allows for increased flexibility in downstream operations, and as a result, reduces the chance of backtracking. Our task planning module receives periodic feedback from the task allocation layer to ensure that the generated plans are compatible with the modules downstream.

At the task allocation level, GRSTAPS assigns agents to tasks based on agent-agnostic partial-order plans from the task planning module. To this end, we contribute a search-based approach that iteratively assigns agents to tasks until the task requirements are satisfied. Notably, our approach to task allocation can simultaneously satisfy task planning constraints and task requirements while optimizing the associated schedule. In order to guide the search, we design two heuristic functions: one that incentivizes allocations that satisfy task requirements, and another that prefers a shorter makespan. Further, our approach to allocation indirectly improves the efficiency of motion plans by accounting for the fact that the feasibility and efficiency of motion plans impact the schedule's makespan.

To evaluate the effectiveness of GRSTAPS and its components, we carry out detailed ablative and comparative experiments on a set of simulated disaster-response problems, generated by varying the number of goal conditions, the number of robots, and the maps associated with the problems. In our ablation studies, we compare GRSTAPS with two se-

quential variants of itself that do not interleave the four modules. In our comparative experiments, we compare our approach with three state-of-the-art temporal planners. Across all our experiments, we find that GRSTAPS outperforms all other approaches in terms of computational efficiency, and scalability with both team size and problem size. Further, when dealing with more than 5 robots, GRSTAPS outperforms all other approaches in terms of solution quality (as measured by requirement satisfaction and schedule makespan).

4.2 Related Work

Reasoning about how a team of heterogeneous robots should individually and collectively interact with the environment is a complex problem that draws elements from several different research fields, including task planning ("*what*"), task allocation ("*who*"), scheduling ("*when*"), and motion planning ("*how*"). While there are numerous works on each individually, the various intersections of the four areas have been attracting increased attention. Solutions to problems that lie at these intersections, although more challenging to solve, provide computational benefits resulting from integrated modeling and reasoning capabilities. In this section, we discuss prior work dedicated to solving problems that lie at intersections of the above listed research areas. We refer readers to comprehensive surveys and taxonomies for more through treatments of these problems and their variants [69, 42, 43, 6, 70, 11, 71, 72].

4.2.1 Multi-Agent Planning

The Multi-Agent Planning (MAP) problem addresses the questions of "*who*" should perform "*what*" tasks "*when*." MAP combines technologies developed by the task planning and multi-agent systems communities. While planning has been generally treated as a single-agent problem, MAP expands upon planning by considering multiple agents that collectively develop a course of tasks to satisfy the goals of the entire group [6]. It also draws elements from task allocation, such as constraints on how many robots can collabo-

rate to execute each task or how many tasks each robot can execute simultaneously.

The MAP problem domain extends the traditional task planning domain with the addition of a finite set of K agents or robots. In this problem, tasks can be executed concurrently, have different durations depending on which robot or coalition of robots is assigned to them, and have complex interdependencies with other tasks based both on time and the agents performing them.

The solution to a MAP problem can be formulated as

- a set with a sequence of tasks for each robot, or
- a partially-ordered plan, which contains a set of tasks and a set of precedence constraints ($\prec$) between tasks in the plan.

There exist a number of state-of-the-art MAP planners that can solve tightly-coupled problems (i.e. planning where agents need to work together to reach goals), such as the Multi-Agent Distributed and Local Asynchronous (MADLA) Planner [73], the Planning State Machine (PSM) [74], and Forward Multi-Agent Planning (FMAP) [75]. In this work, we are particularly interested in a complex variant of the tightly-coupled MAP problem in which agents have to cooperate on individual tasks, or *joint tasks*, such as multiple robots carrying a heavy box. Fewer techniques have the ability to plan for joint tasks in which agents can cooperate on individual tasks [76]. The Partial-Order Multi-agent Planning (POMP) algorithm [77] uses a centralized partial-order planning algorithm based on UCPOP [78] that can handle joint tasks. Multi-Agent Forward Search (MAFS) [79] uses a distributed forward search across a state-space in which individual planning agents exchange information used to determine the heuristic value of each state. Both of these approaches assume that the number of agents and the type of those agents (e.g., two quadcopters) needed to perform each joint task is known *a priori*.

While most of the approaches to solve the tightly-coupled MAP problem with joint tasks require *a priori* knowledge about the exact number and type of agents for each joint task, there are a few approaches that instead solve this problem by placing constraints on the

minimum and maximum number of agents for each joint task. The Single-agent Planning Approach (SPA) for MAP [80] transforms a MAP problem into a single-agent planning problem by splitting tasks, solving the single-agent planning problem with a classical planner, and then merging tasks in their single-agent solution to create joint tasks. Shekhar and Brafman [81] adapt SPA through *collaborative tasks* or single tasks that involve the minimum number of agents necessary to perform a given task. A separate collaborative-task is required for each combination of agents that can execute the task (i.e., for each task that requires two robots, there is a separate instance of the task for each pair of robots). Multi-Agent Planning with Joint Actions (MAPJA) [76] partitions the problem into the loosely-coupled and tightly-coupled components and then solves each separately. It uses an off-the-shelf multi-agent planner to solve the loosely-coupled portion of the problem, and then solves the tightly-coupled portion through calls to a classical planner and incrementally increases the number of agents that can work together to collaborate on each joint task.

The above approaches either require *a priori* knowledge about the exact number and type of agents on each joint task or require constraints on the minimum and maximum number of agents for each joint task. In contrast our framework models tasks with the traits required (e.g., carrying a 10 kg box requires a 10 kg payload). Such trait-based specifications do not require the user to explicitly specify the relationships between each task and robot or coalition, allowing for generalization to different types of agents and various distributions of agent types in coalitions.

4.2.2 Simultaneous Task Allocation and Planning

The Simultaneous Task Allocation and Planning (STAP) problem is a sub-problem under the MAP banner. STAP algorithms aim to solve multi-agent planning problems by constructing a model of the agents and decomposing the plan into parts such that tasks can be allocated to different robots. We specifically highlight this variant of the MAP problem to

illustrate the similarities and differences between STAP and the STAP-STC problem we formulate below in Section 4.3.5.

Multiple prior techniques have proposed the use of Linear Temporal Logic (LTL) to solve the STAP problem. Schillinger *et al.* [82] present and then extend [83] a solution to the STAP problem that works by decomposing the planning problem into sub-tasks while simultaneously allocating these sub-tasks. This decomposition approach means that the algorithm is not forced to search a combinatoric number of allocations when allocating the plan. Chen *et al.* [84] present LTL-based solutions that use trace-closed languages to solve the MAP problem. These trace-based solutions identify individual tasks by projecting the solution onto the agents. Ulusoy *et al.* [85] use a similar approach to [84] but account for infinite horizon task and use a methodology that accounts for the cost of the teams trajectories allowing them to find optimal solutions. Nikou *et al.* [86] offer a Metric Interval Temporal Logic (MITL) approach to solve this problem. This MITL approach allowed agents to have both agent-specific goals and global goals that need to be satisfied by all the team's agents. Faruq *et al.* [87] building on [82] present a algorithm that uses an LTL and MDP-based approach to solve the MAP problem in uncertain environments.

In addition to the LTL-based solutions presented above, others have proposed solutions to the STAP problem that use AI planning techniques and optimization methods. For example, [88] presents distributed heuristic forward search algorithm using a distributed search. Ma and Koenig [89] offer an integer linear programming and conflict-based min-cost-flow algorithm to solve the tamp problem.

These solutions do not allow for coalitions of agents, are unable to plan for tightly-coupled domains, and formulate multi-agent planning differently than GRSTAPS. In contrast to existing STAP algorithms, our framework's trait-based modeling allows for joint tasks (e.g., two robots carrying a 5 kg carry a box that weighs 10 kg). Such trait-based specifications allow for generalization to different types of agents and various distributions of agent types in coalitions.

4.2.3 Task and Motion Planning

The Task and Motion Planning (TAMP) problem combines task planning, which focuses on the high-level discrete decisions about how agents will interact with objects in the environment and what tasks agents will take, with motion planning, which focuses on continuous decisions about paths and the motions of agents [70]. As such, the TAMP problem addresses the questions of "*what*" tasks should be performed "*when*" and "*how*" they should be done. A strictly chained approach in which a task planner chooses a set of tasks then a motion planner finds valid motions for each task is incomplete. There is no guarantee that the tasks selected by the task planner will be geometrically feasible [90]. This causes the key challenge of TAMP to be integrating task planning and motion planning such that it does not break properties or cause practical infeasibility or inefficiencies.

Traditionally, approaches to the TAMP problem solve a single robot variant where the world is fully observable and all tasks are deterministic. In recent years, several approaches have been designed to integrate discrete task planning and continuous motion planning that fall into one of a handful of high-level strategies as we describe below.

Several solutions to the TAMP problem have stemmed from motion planning, including Multi-Modal Motion Planning (MMMP), optimization-based methods, and constraint satisfaction-based methods. Hauser *et al.* [91] introduce MMMP for a system with multiple modes representing different constraint sub-manifolds of the configuration space. They also provide an algorithmic framework for multi-modal planning that utilizes a sampler for transitioning between modes. Barry *et al.* [92] use a bidirectional rrt-style search combined with a hierarchical strategy to suggest tasks on simplified versions of the planning problem. Vega-Brown and Roy [93] extends these ideas to optimal planning with differential constraints. Toussaint *et al.* [94, 95] present a non-linear constrained optimization approach that uses a three-layer logic geometric programming strategy to solve the TAMP problem. Lozano-Perez and Kaelbling [96] present a constraint satisfaction approach to TAMP that uses a symbolic task planner to impose constraints on a geometric motion planner.

Another set of approaches extend symbolic task planning by having the task planner repeatedly query a motion planner. Dornhege *et al.* [97] introduce *semantic attachments* as a way to extend classical planning. Semantic attachments are predicates whose truth values are not established by assertion but by calling an external program utilizing a geometric representation of the state. Similarly, Erdem *et al.* [98] augment a causal reasoning planner to check for the existence of paths for a robot. Kaelbling and Lozano-Perez [99] introduce Hierarchical Planning in the Now (HPN), which is a goal regression-based planner that uses *generators* to perform fast approximate motion planning. Garrett *et al.* [90] present FFRob, a probabilistic complete task and motion planner that utilizes the Fast Forward task planning heuristic and model task conditions as predicates involving geometric and kinematic constraints. Lo *et al.* [100] introduce PETLON, a TAMP algorithm for robot navigation that is optimal at the task level while maintaining manageable computational efficiency.

The third set of approaches interleave task planning and motion planning. Lagriffoul *et al.* [101, 102] focus on limiting the amount of geometric backtracking. A set of approximate linear constraints are imposed by grasp and placement choices, and then linear programming is utilized to compute valid assignments or determine that one does not exist. Srivastava *et al.* [103] interface an off-the-shelf task planner with an optimization-based motion planner. They use a heuristic to remove potentially-interfering objects. Dantam *et al.* [104] formulate TAMP as a Satisfiability Modulo Theorem (SMT) problem. An incremental solver adds motion level constraints to the task level logical formula when a candidate task plan is found. Akbari *et al.* [105] use an ontological knowledge-based task planner based on FF [38] combined with a physics-based motion planner to solve a problem with movable obstacles.

While utilizing different methods for integrating task and motion planning, each of the above approaches is designed for a single robot. More recently, a limited number of techniques to solve the multi-robot variant of the TAMP problem have been proposed. These

multi-robot TAMP approaches extend the work from single robot TAMP approaches presented above. Toussaint *et al.* [106] extend their logic geometric programming framework to solve multi-agent manipulation problems with high-dimensional kinematics. This work, however, is only demonstrated with a single robot and a single human. Shome *et al.* [8] develop a framework for TAMP with multiple arms and anytime behavior. The framework is designed specifically for pick-handoff-place tasks and is not generalizable to other types of tasks or types of robots. Similarly, Umay *et al.* [107] present a two-layer framework for task and motion planning of multiple arms. They use a shared space graph to check whether two arms are sharing certain parts of the workspace. Akbari *et al.* [108] extend their prior work from [105] to multiple robots and allows robots to collaborate on moving certain obstacles. However, in their approach, robots cannot concurrently perform separate tasks, so many of the robots are often waiting idle and the approach only allows homogeneous teams. Motes *et al.* [109] present a conflict based search to task and motion planning for plans with decomposable tasks. This method was capable of planning with 12 robots and 12 tasks; however, this method assumes homogeneous robots.

Prior techniques that address multi-agent TAMP problems are limited in multiple areas. First, all prior work is either limited to a specific problem domain [106, 109] or a particular type of robot [8, 107, 108]. Additionally, none of these approaches can handle tightly-coupled problems with joint tasks. In contrast, our framework is domain-independent, can handle heterogeneous teams of robots, and can solve tightly-coupled problems with joint-tasks.

4.2.4 Multi-Vehicle Routing

The Multi-Vehicle Routing (MVR) problem requires selecting a set of vehicles and their routes to visit a set of locations [7]. This problem addresses the questions of "*who*" should visit each location and "*how*" these locations should be reached. MVR is a sub-problem under the larger banner of Multi-Agent Motion Planning (MAMP) which involves finding

collision-free trajectories connecting each agent's initial location to its goal location [110, 111, 112]. We limit our coverage of related work specifically to MVR as it is our most closely-related variant of MAMP and shares many of the assumptions made by our proposed solution.

There exists a large number of solutions to the different variants of this problem. In this work, we are particularly interested in the subset that also deals with temporal constraints. Bredstrom *et al.* [9] add synchronization constraints to the Multi-Vehicle Routing Problem with Time Windows (MVRP-TW) and then solves the problem using an optimization-based heuristic. Ramchurn *et al.* [54] use a Mixed-Integer Linear Programming (MILP) formulation to solve a problem they call the Coalition Formation with Spatial and Temporal Constraints Problem (CFSTP). Jones *et al.* [46] present two methods to solve an MVR problem with intra-path and temporal constraints. The first uses a tiered auction with two heuristics to conduct a bounded search of possible schedules, allocations, and routes. The second uses a centralized genetic algorithm for better quality solutions but at the cost of significantly more computation. Korsah *et al.* [50] solve a variant of the MVR problem with cross-schedule temporal constraints using a custom branch-and-price algorithm. Flushing *et al.* [113] use a MILP formulation in conjunction with a two-layered meta-heuristic comprised of a genetic algorithm and an iterative local search.

In the presented MVR solutions, agents are modeled in terms of the tasks they can be assign to and the cost of transitioning nodes on the graph. Furthermore, in each approach, the number and type of agents assigned to each task are known *a priori*. In contrast, GRSTAPS uses a trait-based framework that allows for a more robust model of agent capabilities than existing approaches. This more robust modeling allows for GRSTAPS to better represent the heterogeneity of agents and allows for a more thorough description of task requirements when compared to existing approaches.

4.3 Problem Formulation

In this section, we formalize a class of problems that we refer to as Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC). STAP-STC takes a holistic view of heterogeneous multi-robot coordination by simultaneously considering task planning, allocation, scheduling, and motion planning. We begin by describing the formulation of each of the sub-problems that STAP-STC encompasses and finish by describing aspects that are specific to the STAP-STC problem.

4.3.1 Temporal Planning

In temporal planning, the state of the world is represented by a set of first-order literals (e.g., Damaged(Building1), OnFire(Building1)), and fluents (e.g., LocationOf(Box1)), each of which describes some condition that holds in the world. Most temporal planners, GRSTAPS included, operate under a closed-world assumption. Goals are defined similarly to states, and a goal is said to be satisfied by a world state if each of its literals explicitly appears in the state description, and each of its fluents is assigned the specified value.

PutOutFire(b - building):
dur:30
cond:
OnFire(b)
eff:
¬ OnFire(b))

RepairBuilding(b - building):
dur:600
cond:
¬ OnFire(b)
¬ UnderRubble(b)
Damaged(b)
eff:
¬ Damaged(b))

Figure 4.1: Example Task Schemas

Tasks, also known as actions in some of the temporal planning literature, are represented as a tuple that contains the duration of the task as a continuous value, a set of conditions that must hold for the task to be applicable, and a set of effects that transform the world state by asserting or retracting certain literals. These are collected together in a Task Schema, such

as shown in Figure 4.1, which is defined in terms of parameters that are instantiated during the planning process. For example, the PutOutFire task shown in Figure 4.1 can only be applied to the entity b if b is a building and is on fire, and has the effect of retracting the literal OnFire(b) from the world state.

A specific temporal planning problem is defined by an initial world state s_{init}, a goal G, and finite set of tasks $\mathcal{T}$. The planner's objective is to find a plan that transitions the initial state s_{init} to a state in which the conditions of the goal G are satisfied. A partial-order plan $\pi = \langle \mathrm{T},\ \mathcal{P},\ \mathcal{L} \rangle$ is a tuple where $\mathrm{T} \subseteq \mathcal{T}$ is a set of tasks; $\mathcal{P}$ is a set of precedence constraints ($\prec$) between tasks in T; and $\mathcal{L}$ is a set of causal links between tasks in T. A **precedence constraint** $\tau_i \prec \tau_j$ is a temporal relationship between two tasks, τ_i and τ_j, that ensures that the execution of τ_i concludes before τ_j starts [30]. A **causal link**, denoted by $\tau_i \xrightarrow{p} \tau_j$, indicates that precondition p of τ_j is supported by one of the effects of τ_i. A partially-ordered plan can be represented by a directed graph in which each vertex represents a task and each edge, $e = \tau_i \rightarrow \tau_j$, represents an ordering constraint ($\tau_i \prec \tau_j$). A **task network** is a graph where each vertex represents a task and each edge represents a temporal constraint. Therefore, a partial-order plan is a specific type of task network that only has precedence constraints as edges.

4.3.2 Trait-based Time-extended Task Allocation

We consider a heterogeneous team of K robots that must collectively execute a set of tasks. Each robot is defined by its abilities or *traits* [5, 114, 40], which are modeled as continuous variables, encoded by the trait vector

$$\mathbf{q}^{(n)} = \left[\mathbf{q}_1^{(n)},\ \mathbf{q}_2^{(n)},\ \cdots,\ \mathbf{q}_U^{(n)} \right] \tag{4.1}$$

in which $\mathbf{q}_i^{(n)} \in \mathbb{R}_{\geq 0}$ corresponds to the i^{th} trait for the n^{th} robot (e.g. a robot with a payload capacity of 10 kg and a speed of 5 m/s might have trait vector $\mathbf{q} = [10, 5, 0, 0]$ if

there are four relevant traits for the problem domain, the first two being payload capacity and speed). The traits of the entire team are defined by the **robot trait matrix**

$$\mathbf{Q} = \begin{bmatrix} \mathbf{q}^{(1)} \\ \vdots \\ \mathbf{q}^{(K)} \end{bmatrix} \in \mathbb{R}_{\geq 0}^{K \times U} \tag{4.2}$$

with each row corresponding to one robot and each column corresponding to one trait. This trait-based modeling of agents allows for a robust description of each agent and makes the heterogeneity of agents easily interpretable.

Each task in the task network T may be executed individually or collectively as part of a coalition, depending on its trait requirements (e.g., picking up a given box may require one robot or multiple robots depending on payload capabilities). The traits required for an individual task τ_i are defined by a **task trait requirements vector**

$$\mathbf{y}^{(i)} = \left[\mathbf{y}_1^{(i)}, \ \mathbf{y}_2^{(i)}, \ \cdots, \ \mathbf{y}_U^{(i)} \right] \tag{4.3}$$

in which $\mathbf{y}_u^{(i)} \in \mathbb{R}_{\geq 0}$ is the u^{th} trait requirement (e.g. if moving a survivor to a hospital requires a coalition with a 1 kg payload capacity and a speed of 2 m/s and does not require any water or construction ability would have a task trait requirements vector $\mathbf{y} = [1, 2, 0, 0]$).

The traits required by the entire task network T are defined by the **desired trait matrix** $\mathbf{Y}^T$ where:

$$\mathbf{Y}^T = \begin{bmatrix} \mathbf{y}^{(1)} \\ \vdots \\ \mathbf{y}^{(|T|)} \end{bmatrix} \in \mathbb{R}_{\geq 0}^{|T| \times U} \tag{4.4}$$

in which $\mathbf{y}^i$ is the task trait requirements vector for the i^{th} task in T and $|T|$ represents the number of tasks in T.

A trait-based time extended task allocation problem is defined by a task network T, a robot trait matrix $\mathbf{Q}$, and a desired trait matrix, $\mathbf{Y}^T$, for the given task network.

The assignment of agents to tasks can be represented by an **allocation** $\mathbf{A}$ where:

$$\mathbf{A} = \begin{bmatrix} a_{1,1} & \cdots & a_{1,k} \\ \vdots & \ddots & \vdots \\ a_{n,1} & \cdots & a_{n,k} \end{bmatrix}$$

where

$$a_{n,k} = \begin{cases} 1 & \text{if the } k^{th} \text{ robot is assigned to the } n^{th} \text{ task} \\ 0 & \textit{otherwise} \end{cases}$$

The solution to a trait-based task allocation problem is an assignment of robots to tasks in the task network. An allocation $\mathbf{A}$ satisfies $\mathbf{Y}^T$ when $\mathbf{AQ}$ is element-wise greater than or equal to $\mathbf{Y}^T$ (See Figure 4.2).

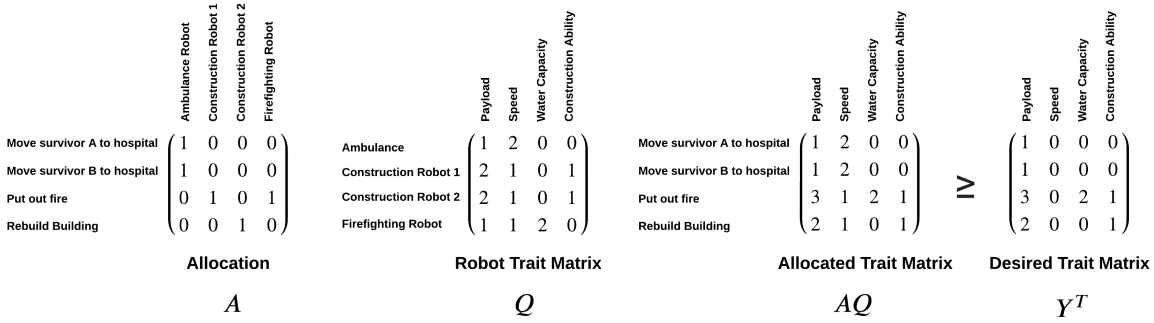


Figure 4.2: An example of $\mathbf{AQ}$ compared to the desired traits matrix $\mathbf{Y}^T$. As $\mathbf{AQ}$ is element-wise greater than or equal to the desired traits matrix, it satisfies the requirements.

4.3.3 Scheduling

The scheduling problem involves the determination of when tasks begin and end. A schedule σ is an assignment of start and end times to each task in a task network. Additionally, a valid schedule must respect all temporal constraints imposed by the task network (e.g.,

precedence constraints or deadlines), all temporal constraints imposed by the allocation (e.g., mutex constraints), and all temporal constraints imposed by the durations of motion plans (e.g. transition constraints). A **mutex constraint** is a relationship between two tasks, τ_i and τ_j, that ensures that either τ_j must finish before τ_i starts or τ_i must finish before τ_j starts [14]. These mutex constraints are created when a robot is assigned to a task τ_i, making it unable to perform another task τ_j at the same time as τ_i. A **transition constraint** is a relationship between two tasks, τ_i and τ_j, that ensures that τ_j does not start until after a certain time after τ_i finishes. If a robot is supposed to execute τ_i and then τ_j, it must travel from the location where it finished τ_i to the location where it is supposed to execute τ_j. As such, transition constraints help ensure that there is sufficient time for inter-task travel.

4.3.4 Motion Planning

The motion planning problem (see, e.g., [57]) is to find collision-free paths for the robots as they perform the tasks in Task Network T under the Allocation $\mathbf{A}$ and the Schedule σ. The world W describes all of the static geometric information about the environment that the robots will need to navigate including obstacles.

For an individual robot, the collision-free path can be represented as a continuous map $\gamma : [0, 1] \rightarrow \mathcal{C}$, through the configuration space such that the robot does not collide with the obstacles in W, dynamic obstacles, other robots, or movable objects. A configuration is a point in the configuration space and the configuration of a robot at any given time, represents its geometric state at that time. For example, configuration can include the (x, y, z) coordinates, orientation, steering angle, velocity, or other features that characterize the robot. Note that the individual configuration spaces for different robots can be qualitatively different, e.g., a quadcopter can fly over obstacles, while a ground vehicle cannot.

For Task τ_i, we define $\mathcal{C}_{\tau_i}^{init}$ and $\mathcal{C}_{\tau_i}^{term}$ as the joint initial and terminal configuration spaces of the robots assigned to τ_i. For these robots to successfully preform Task τ_i, they must first find collision-free paths from their current configurations to a configuration in

$\mathcal{C}_{\tau_i}^{init}$ and then after starting the task they must find collision-free paths to a configuration in $\mathcal{C}_{\tau_i}^{term}$. As such, $\mathcal{C}_{\tau_i}^{init}$ and $\mathcal{C}_{\tau_i}^{term}$ can be viewed as specifying geometry-based pre- and postconditions of Task τ_i.

The Multi-Agent Motion Planning problem is defined by the initial configurations of all robots I_c, the set of initial and terminal configurations for all tasks $\mathcal{C}_T$, and a world W which describes all of the static geometric information about the environment. For a given task network T under the allocation $\mathbf{A}$, a solution to the motion planning problem is given by a set of motion plans X, that allows each robot to complete each of its assigned tasks, including paths that transition between sequential tasks.

The problem becomes slightly more complex for the case of multi-agent robot teams executing tasks in a task network. First, the free configuration space for an individual robot may not be static during the execution of a task. Thus, a motion planner should take into account the possibility of inter-robot collision as well as the possibility of collision with static obstacles. Second, because objects in the world can change location during task execution, a robot's free configuration space while executing task τ_i may depend on the order in which tasks are executed. This particular problem gives rise to much of the complexity of the TAMP problem.

4.3.5 Simultaneous Task Allocation and Planning with Spatiotemporal Constraints

We now formalize the *Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC)* domain. STAP-STC problems take a holistic view of heterogeneous multi-robot coordination by simultaneously considering all of the problems discussed in the previous sections. This merging of task planning, motion planning, scheduling, and trait-based Time-Extended task allocation creates a more complete view of all aspects of the multi-robot coordination problem.

A STAP-STC problem is defined by

- s_{init}, the symbolic initial state for the task planner;

- G, the symbolic goal for the task planner;
- $\mathcal{T}$, the finite set of all tasks;
- $\mathbf{Q}$, the robot trait matrix;
- I_c, the set of initial robot configurations, with one for each robot;
- $\mathcal{C}_{\mathcal{T}}$, the set of initial and terminal configuration spaces for each task in $\mathcal{T}$;
- and W, the world describing all of the static geometric information about the environment.

For a specific problem, we wish to compute the solution $\left\langle \hat{\pi}, \hat{\mathbf{A}}, \hat{\sigma}, \hat{X} \right\rangle$ where:

- $\hat{\pi}$ is a partially-ordered plan that transitions the world to a symbolic state the satisfies the set of goal conditions G,
- $\hat{\mathbf{A}}$ is an allocation that satisfies the desired trait matrix of $\hat{\pi}$ ($\mathbf{Y}^{\hat{\pi}}$),
- $\hat{\sigma}$ is a schedule that satisfies the temporal constraints created by $\hat{\pi}$ and $\hat{\mathbf{A}}$,
- and $\hat{X}$ is a set of discrete trajectories, one for each robot. As part of these trajectories, each robot moves to the tasks it was allocated in the order they were scheduled while avoiding collisions with other robots and the environment. The other trajectories in this set represent the movements needed to execute the tasks.

4.4 Communication Between Sub-Problems

Our approach involves solving multiple sub-problems and to reduce computation, we exploit the fact that as each sub-problem is solved constraints are generated on and between the same shared components (i.e., tasks and robots) and these constraints can be shared between the layers working on each sub-problem. This approach utilizes both *positive constraints*, which are constraints based on what *can* be satisfied, and *negative constraints*, which are constraints based on what *cannot* be satisfied. Methods that create positive constraints usually have a a database of constraints paired with their satisfying assignment. This is sometimes known as memoization. Alternatively, methods that create negative con-

straints usually focus on identifying counterexamples.

Additionally, for our approach we expand constraints to their most general context. For example, if we determine that a specific robot cannot contribute to the execution of a task because its traits cannot contribute, it is possible that that constraint of "robot x cannot contribute to task τ_y" can be generalized to "species s cannot contribute to task τ_y" or if we find a motion plan between two configurations for a specific robot it can be utilized by other robots of the same specifies. Furthermore, if we independently we determine that none of the species for a given problem can contribute to a task, we can conclude that the task cannot be accomplished at all and task planning can utilize this constraint to reduce its branching factor. More constraints and generalizations will be discussed in the next section.

4.5 Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling

This section outlines the high-level algorithmic architecture used for Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling (GRSTAPS). First, we introduce each of the layers of the framework at a high level, provide a running example, discuss the specific assumptions made by our approach, and then explain each layer in more detail.

The top layer of the GRSTAPS framework is the task planning layer. This layer incrementally builds a plan $\hat{\pi}$ that transitions the world to a state that satisfies the set of goal conditions G. The layer builds $\hat{\pi}$ by starting with an empty plan π_0 and adding a single task at a time. During the addition of each task, the planning layer ensures that constraints, such as preconditions and postconditions, continue to be satisfied. Thus, each addition of a task and ordering constraints creates a new plan π.

When the task planning layer creates a new plan $\grave{\pi}$, it passes $\grave{\pi}$ to the task allocation layer. The task allocation layer attempts to build an allocation of robots for each task $\tau \in \mathcal{T}$ in $\grave{\pi}$. In order to build $\hat{\mathbf{A}}$, the task allocation layer conducts a graph search through a space of task allocations.

When the task allocation layer creates an allocation $\grave{\mathbf{A}}$ for a plan $\grave{\pi}$, it passes both $\grave{\mathbf{A}}$ and $\grave{\pi}$ to the scheduling layer. The scheduling layer determines if a temporally consistent schedule $\grave{\sigma}$ can be created from $\grave{\mathbf{A}}$ and $\grave{\pi}$. While building the schedule, the scheduling layer queries the motion planner for any required motion plans. The scheduling layer uses information from the motion plans to adjust the start and endpoints for each task in $\grave{\pi}$ to account for the traversal time of agents.

Our framework iterates between the layers until it finds a solution that satisfies all of the problem's constraints. This hierarchy of nested algorithms solve the problem with varying abstractions for effective and efficient planning, allocation, and scheduling.

We will utilize a running example based on a disaster recovery scenario to illustrate how the GRSTAPS framework solves the STAP-STC problem. In this example, two wounded survivors need to be taken to a hospital for treatment and a burning building. The goal for this problem is that both survivors have been brought to a hospital, the fire has been extinguished, and the building has been repaired. A team of robots, including two construction bots, a firefighting robot, and a paramedic robot, will be used to solve the problem. We demonstrate how GRSTAPS selects a set of appropriate tasks, assigns these tasks to coalitions of robots, schedules the tasks, and provides motion plans for all robots so they can execute the plan. We will revisit this example throughout the remainder of this section, including Figures 4.3-4.5.

4.5.1 Algorithmic Assumptions

Given the complexities of the general STAP-STC problem, GRSTAPS makes several assumptions regarding the structure of the STAP-STC formulation. In the interest of clarity and completeness, we list the specific assumptions below:

- All tasks and goal conditions are agent-agnostic (i.e., all task preconditions and effects are specified in the context of the symbolic environment and not the context of the robots).

- The environment is deterministic and is only changed as a function of the robots executing tasks.
- All tasks are grounded by the task planning layer of GRSTAPS.
- Robot capabilities can be modeled by a vector of continuous traits that do not change throughout execution.
- Each grounded task is associated with a vector of continuous trait requirements that do not change throughout execution.
- The set of initial and terminal configuration spaces for each task in $\mathcal{T}$, $\mathcal{C}_{\mathcal{T}}$, is known *a priori*.
- The free configuration space of each robot is static and does not change when robots move objects in the environment. It is assumed that modern motion controllers can handle avoiding collisions with moved objects [82, 87, 100].
- Motion planning only considers collisions with the environment when planning paths for robots and collisions. It does not consider robot on robot collisions. It is assumed that modern motion controllers can handle avoiding collisions with other robots [82, 87, 100].

In Section 7.2, we discuss some of the limitations of these assumptions and how future work can address them.

4.5.2 Task Planning

The task planning layer requires an incremental partial-order planner. For an explanation on the specifics of how a traditional partial-order planner solves a task planning problem and some specific partial-order planning terminology, we refer the reader to Section 2.2. This task planner starts with an empty plan π_0. It then incrementally adds a single task τ_i and constrains the ordering ($\prec$) of the new task with respect to the tasks already in the plan. Each task is agent-agnostic, and so the task planner does not consider which robot or coalition is supposed to execute it. This addition of a task and ordering constraints creates a

new plan $\grave{\pi}$. The plan $\grave{\pi}$ is a valid partial-order plan with no threats or open conditions (see Section 2.2 for definitions); however, it does not necessarily transition the world to a state that satisfies the set of goal conditions G. A plan that transitions the world to a state that satisfies G will be denoted as $\hat{\pi}$ for simplicity; however, one should note that multiple plans may transition the world to a state that satisfies G and its possible that there are multiple world states that satisfies G.

Then, a *desired traits matrix* $\mathbf{Y}^{\grave{\pi}}$ is created for $\grave{\pi}$ by concatenating the *task traits requirements vectors* associated with each task in $\grave{\pi}$ (see Equations 4.3 and 4.4).

The plan $\grave{\pi}$ and its associated desired traits matrix $\mathbf{Y}^{\grave{\pi}}$ are passed to the task allocation layer. In conjunction with the scheduling and motion planning layers, the task allocation layer determines if a feasible allocation, schedule, and set of motion plans can be found for $\grave{\pi}$ that satisfies the various constraints of the problem. If they cannot be found, then $\grave{\pi}$ is pruned, and other plans are considered. If is possible that collectively task allocation, scheduling, and motion planning discover constraints that conjunctively general to a specific task cannot be executed by the robots available. The task planning layer can utilize these constraints to reduce its branching factor as it continues the search.

If an allocation $\hat{\mathbf{A}}$, a schedule $\hat{\sigma}$, and a set of motion plans $\hat{X}$ can be found for a plan $\hat{\pi}$, then $\left\langle \hat{\pi}, \hat{\mathbf{A}}, \hat{\sigma}, \hat{X} \right\rangle$ is a solution to the problem.

FCPOP

In this work, the task planning layer uses a modified version of the Forward Chaining Partial-Order Planner (FCPOP) planner [63] described in Chapter 2. FCPOP combines traditional partial-order planning with grounded forward chaining for robust and fast domain-independent temporal planning. FCPOP removes the delayed parameter binding of partial-order planning by grounding the tasks before the search but fully utilizes the delayed task ordering commitment from traditional partial-order planning. This allows tasks to be added to any point in the plan. As such, the forward search can be viewed as a commitment to a

set of tasks and not to the order of their application [37]. Furthermore, FCPOP interleaves information between task planning and scheduling layers. For GRSTAPS we utilize the task planning portion of the original FCPOP on agent-agnostic tasks, but instead of the previous scheduling layer described in Chapter 2, we use the scheduling layer described below.

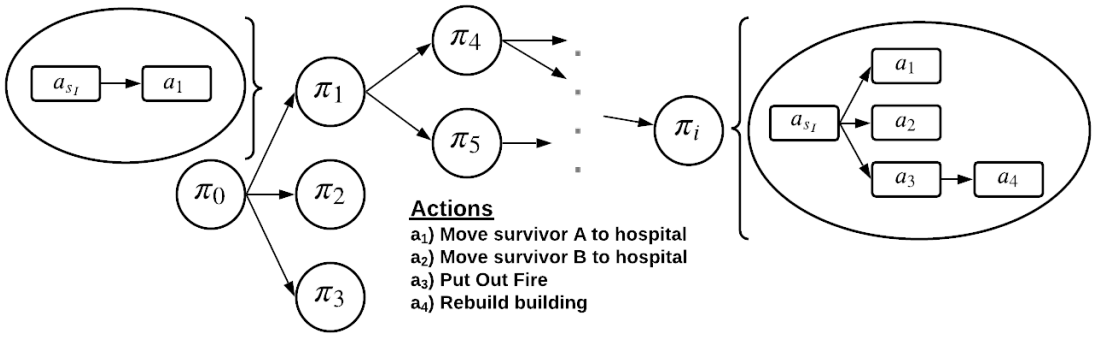


Figure 4.3: An example of the task planning search. The search starts from a plan with a single dummy task and searches to through a plan space to find a solution plan

As with the traditional FCPOP, the temporal planner in GRSTAPS performs an A* search through plan space in which each node of the tree represents a partially-ordered plan (see Algorithm 6). Figure 4.3 illustrates this search through plan space. Prior to the search, the planner creates fictitious tasks $\tau_{s_{init}}$, whose effects assign the values needed to satisfy the conditions in the initial state s_{init}, and τ_G, whose conditions are the set of goal conditions G. The search starts at the root node which contains a partial-order plan $\pi_0 = \langle\{\tau_{s_{init}}\}, \emptyset, \emptyset\rangle$ with the fictitious task $\tau_{s_{init}}$ and no causal links or ordering constraints. Then when the planner expands a node $\grave{\pi}$, it determines which tasks can be added to $\grave{\pi}$. A task is only feasible to add if, for each condition in the new task, a causal link can be created from an effect of one of the tasks currently in $\grave{\pi}$ to support the condition. It is possible that the addition of the new task τ_k creates a threat to a causal link between two tasks, τ_i and τ_j, that are already in the plan over τ_j's precondition p ($\tau_i \xrightarrow{p} \tau_j$). A threat is created when one of the effects of the new task τ_k modifies the value of a precondition away from the value

Algorithm 6: Task Planning Layer

Input: $\langle s_{init},\ G,\ \mathcal{T} \rangle$

Output: $\left\langle \hat{\pi},\ \hat{\mathbf{A}},\ \hat{\sigma}, \hat{X} \right\rangle$

$\tau_{s_{init}} \leftarrow$ fictitious action for the initial state
$root \leftarrow \langle \{\tau_{s_{init}}\}, \emptyset, \emptyset \rangle$
$pq \leftarrow$ PriorityQueue($\{root\}$)
while *pq is not empty* **do**
    // Pop off the current best plan node
    $n \leftarrow pq$.pop()
    // Check if the node is a solution
    if *n contains a solution* **then**
        return $n.\pi, n.\mathbf{A},\ n.X,\ n.\sigma$
    for *task* $\tau \in \mathcal{T}$ **do**
        if *τ is applicable in $n.\pi$* **then**
            $n' \leftarrow$ create new node
            $n'.\pi \leftarrow$ apply τ to n.π
            $\mathbf{Y}^{n'.\pi} \leftarrow$ create desired traits matrix from $n'.\pi$
            $n'.\mathbf{A},\ n'.\sigma,\ n'.X \leftarrow$ TaskAllocationLayer($n'.\pi$, $\mathbf{Y}^{n'.\pi}$)
            // Generate the frontier state by simulating the effects of $n.\sigma$
            $s_{fr} \leftarrow$ generate frontier state from $n.\sigma$
            // The path cost for a node is the makespan of the generated schedule
            $n'.g \leftarrow C_{n'.\sigma}$
            // The heuristic value is generated by FCPOP's variant of the TRPG
            $n'.h \leftarrow$ TRPG($s_{fr},\ G,\ \mathcal{T}$)
            pq.push(n')
return *null*

needed for the causal link, and there is no temporal relationship between τ_k and either τ_i or τ_j that would prevent it from doing so. If adding the new task τ_k creates a threat then the planner determines whether that threat can be resolved through either promotion ($\tau_k \prec \tau_i$) or demotion ($\tau_j \prec \tau_k$). If the task can be added and any threats can be resolved, this creates a new plan. An example of this can be found in Figure 4.3, in which the root node π_0 is expanded and its child node π_1 has the task τ_1 added.

If the temporal planner in GRSTAPS were to use the traditional heuristic process of creating a Simple Temporal Network (STN) [30] from FCPOP's original scheduling layer, computing the frontier state, and then generating its variant of the Temporal Relaxed Plan-

ning Graph (TRPG), then the solution plan would be inaccurate for all but the trivial problem where none of the robots move as it does not consider the travel times for the robots or any other temporal constraints from the assignment of the robots to tasks. Instead, it uses an STN to check for temporal consistency of a task plan $\grave{\pi}$. If $\grave{\pi}$ is temporally consistent, then the task planning layer passes $\grave{\pi}$ and the desired traits matrix $\mathbf{Y}^{\grave{\pi}}$ to the task allocation layer. We describe the specifics of the task allocation, scheduling, and motion planning layers below, but at a high level, one of the things that they produce is a schedule $\grave{\sigma}$ for the tasks in $\grave{\pi}$ based on temporal constraints from$\grave{\pi}$, temporal constraints from an allocation $\grave{\mathbf{A}}$ of robots to the tasks in $\grave{\pi}$, the travel times for robots to execute each task they are assigned, and the time for each robot to travel to the initial configuration for each task it is assigned. The makespan of this schedule is then used as the path cost for $\grave{\pi}$, and the schedule is simulated to create a frontier state used by the TRPG variant. The task planner then continues its search.

For our example, the task planner passes π_i from Figure 4.3 to the task allocation layer.

4.5.3 Task Allocation

In this work, the task allocation layer uses the Incremental Task Allocation Graph Search (ITAGS) algorithm [5] as described in Chapter 3 due to its demonstrated efficiency. It should be noted, however, that the GRSTAPS framework can use any algorithm designed to solve the time-extended trait-based task allocation problem as presented in Section 4.3.2. The task allocation layer makes an allocation $\mathbf{A}$ of robots for each of the tasks in a plan π that satisfies the desired traits matrix $\mathbf{Y}^{\pi}$. An allocation $\mathbf{A}$ satisfies $\mathbf{Y}^{\pi}$ when $\mathbf{AQ}$ is element-wise greater than or equal to $\mathbf{Y}^{\pi}$ (See Figure 4.2).

It receives from the task planner the plan $\grave{\pi}$ and the desired traits matrix $\mathbf{Y}^{\pi}$. The task allocator performs a greedy best-first search through the incremental task allocation space (See Algorithm 7). In this space, a node represents an allocation of robots to tasks. Nodes are connected to other nodes that differ only by the assignment of a single robot. This

Algorithm 7: Task Allocation Layer

Input: $\langle T,\ \mathbf{Y}^T \rangle$
Output: $\langle \mathbf{A},\ X,\ \sigma \rangle$

$\mathbf{Q} \leftarrow$ from problem description
$I_c \leftarrow$ from problem description
$\mathcal{C}_T \leftarrow$ from problem description
$\alpha \leftarrow$ user defined parameter
$root \leftarrow$ empty allocation
$pq \leftarrow$ PriorityQueue($\{root\}$)
while *pq is not empty* **do**
    // Identify the node with lowest heuristic value
    $n \leftarrow pq$.pop()
    // Check if the node is a solution
    if $n.apr$ == *0* ***and*** $n.nsq < \infty$ **then**
        return $n.\mathbf{A},\ n.X,\ n.\sigma$
    // Compute heuristics for each successor
    for $n' \in$ *generateSuccessors(n)* **do**
        // Compute using Equation 4.5
        $n'.apr \leftarrow$ APR(child, $\mathbf{Q}$, $\mathbf{Y}^*$)
        // Compute schedule and motion plan
        $\sigma,\ \sigma_{LB},\ \sigma_{UB},\ X \leftarrow$ schedule(n', T)
        // Compute using Equation 4.6
        $n'.nsq \leftarrow$ NSQ($\sigma,\ \sigma_{LB},\ \sigma_{UB}$)
        // Compute using Equation 4.7
        $n'.tetaq = \alpha\ *\ n'.apr + \ (1\ -\ \alpha)\ *\ n'.nsq$
        pq.push(n')
return *null*

graphical representation allows our search to start from an initial node with no allocated robots and incrementally add robots until an allocation satisfies the desired traits matrix for a plan.

Figure 4.4 illustrates the search of the incremental task allocation graph for π_i from our example. For $\mathbf{A}_k$ in Figure 4.4, the paramedic robot is assigned to independently move both survivors A and B to the hospital, construction robot 2 and the firefighting robot are assigned to collaboratively put out a fire, and construction robot 1 is assigned to repair a building.

To guide the search, we use two heuristics: *Allocation Percentage Remaining*, which

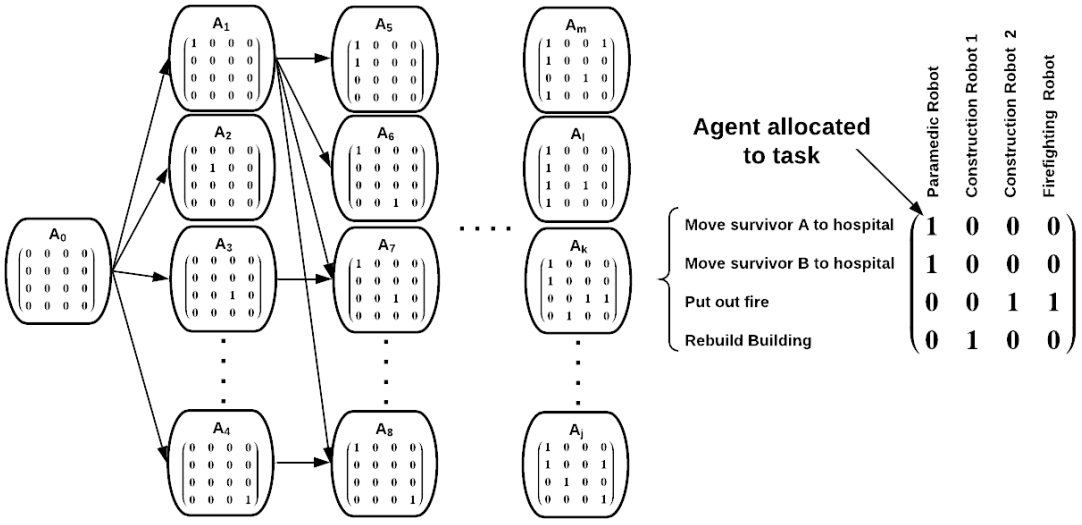


Figure 4.4: The associated incremental task allocation graph for plan $\pi_{\mathbf{i}}$ from Figure 4.3

guides the search based on the quality of the allocation, and *Normalized Schedule Quality*, which guides the search based on the quality of the makespan of the schedule associated with the allocation. We use a convex combination of the two heuristics, which we call *Time-Extended Task Allocation Quality*. The following sections will discuss these heuristics in detail, and Algorithm 7 contains the pseudo-code for the task allocation layer.

Allocation Percentage Remaining

Allocation Percentage Remaining (APR) is defined as the percentage trait mismatch error. Specifically, APR is calculated as

$$f_{apr}(\grave{\mathbf{A}}) = \frac{||\max(\mathbf{Y}^{\grave{\pi}} - \grave{\mathbf{A}}\ \mathbf{Q},\ 0)||_{1,1}}{||\mathbf{Y}^{\grave{\pi}}||_{1,1}} \tag{4.5}$$

in which $\grave{\mathbf{A}}$ is the current allocation that the heuristic is evaluating. This equation sums the error between the desired traits for each task ($\mathbf{Y}^{\grave{\pi}}$) and the traits provided by the coalition that is assigned to execute that task. The error is then normalized by the element-wise summation of the desired traits matrix $\mathbf{Y}^{\grave{\pi}}$.

APR does not use any information from the scheduling layer and, as such, tends to search the graph deeply. This behavior is caused by nodes deeper in the graph having more robots assigned and a smaller desired trait mismatch error. This deep search leads APR to find an allocation that quickly satisfies the desired traits matrix at the expense of ignoring schedules with considerably shorter makespans.

Before each task allocation search, we can quickly calculate the APR when all robots are assigned to all tasks. If the value is not zero, then the most recently added task cannot be allocated and the task planning layer can utilize this information to not only prune this specific task plan, but reduce its branching factor going forward as it can remove the task from the full set of grounded tasks. Additionally, upon the first expansion from the root node, we can quickly determine which species of robots cannot contribute to each task. These constraints decrease the branching factor for the remainder of the current task allocation search and for all future task allocation searches conducted for other partial-order plans as part of solving the entire problem, increasing efficiency. Furthermore, it is possible that a conjunction of constraints created by the task allocation, scheduling, and motion planning layers generalize to "there are no combination of robots that can execute the most recently added task." This information is passed up to the task planning layer, where once again it can be utilized to prune the current partial-order plan and reduce the set of all grounded tasks which in turn reduces the branching factor of the task planning search. This conjunctive reasoning would occur on the expansion of the root node of the task allocation search causing the task planning layer to efficiently move on to another partial-order plan.

Normalized Schedule Quality

The second heuristic, Normalized Schedule Quality (NSQ) is a measure of how much an allocation minimizes the makespan of its accompanying schedule. Specifically, NSQ is calculated as

$$f_{nsq}(C_{\grave{\sigma}}) = \frac{C_{\grave{\sigma}} - C_{\sigma_{LB}}}{C_{\sigma_{UB}} - C_{\sigma_{LB}}} \tag{4.6}$$

in which C_σ is the makespan, or completion time, of the schedule σ, $\grave{\sigma}$ is the schedule based on $\grave{\pi}$ and $\grave{\mathbf{A}}$, σ_{LB} is the schedule without any constraints placed on the schedule from the allocation and motion planning, and σ_{UB} is the schedule in which the plan fragment is totally-ordered and all motion plans are assumed to be the longest possible length. The longest possible length path was calculated as the sum of the perimeter of every obstacle in the environment as well as the perimeter of the environment's bounding box.

The three variables $\grave{\sigma}$, σ_{LB}, and σ_{UB} are all computed by the scheduling layer. As NSQ only considers the schedule and not the allocation, it tends to favor a broader search. This broad search is caused by nodes closer to the root having fewer temporal constraints, more concurrency, and lower makespans. This broad search leads to NSQ finding an allocation that satisfies the desired traits matrix with the minimum makespan at the expense of searching a much larger area of the graph.

Time-Extended Task Allocation Quality

To balance the shortcomings of these heuristics we use a convex combination of APR and NSQ which we call Time-Extended Task Allocation Quality (TETAQ), this heuristic considers both the quality of the allocation as well as the quality of the schedule. The equation for TETAQ is as follows

$$f_{tetaq}(\grave{\mathbf{A}}, C_{\grave{\sigma}}) = \alpha f_{nsq}(C_{\grave{\sigma}}) + (1-\alpha) f_{apr}(\grave{\mathbf{A}}) \tag{4.7}$$

In this equation, $\alpha \in [0, 1]$ represents a user-specified parameter that weights the relative influence of APR and NSQ. If $\alpha = 0$ then this heuristic simplifies to APR, and if $\alpha = 1$ then this heuristic simplifies to NSQ. TETAQ uses α to take qualities from both to perform the search, which allows it to balance finding an allocation that satisfies the desired traits quickly with finding an allocation that minimizes the makespan of the assigned robots' schedule.

For our example, the task allocation layer passes π_i and $\mathbf{A}_k$ to the scheduling layer.

4.5.4 Scheduling

In this work, the scheduling layer uses the the scheduling layer from our ITAGS algorithm [5] as described in Chapter 3 due to the demonstrated efficiency of ITAGS. It should be noted, however, that the GRSTAPS framework can use any algorithm designed to solve the scheduling problem as presented in Section 4.3.3.

The scheduling layer determines if it is feasible to create a schedule for the tasks in a plan while ensuring that all temporal constraints imposed by the task planner and task allocator are satisfied. This layer receives a plan $\grave{\pi}$ and an allocation $\grave{\mathbf{A}}$ from the task allocation layer.

The scheduler finds a schedule σ associated with $\grave{\pi}$ and $\grave{\mathbf{A}}$, which is then used as part of the NSQ heuristic for the task allocation layer. Also, the makespan of the schedule $C_{\grave{\sigma}}$ is used as the path cost of the task planning node for $\grave{\pi}$. When building a schedule, the scheduler must consider three temporal components:

- The static duration of each task.
- The time needed for each robot to travel to the task's initial configuration.
- The time needed for the assigned coalition of robots to execute the movements required by the task.

To this end, the scheduler provides each task's initial and terminal configurations to the motion planning layer to determine if there is a feasible motion plan between the two configurations. The scheduler then uses either the robot or the coalition's speed to determine the time needed to execute the motion plan. Using this information the schedule then calculates $\grave{\sigma}$, σ_{LB}, and σ_{UB}.

To find a schedule for a plan and an allocation, the scheduler must handle three different types of temporal constraints:

- precedence constraints imposed by the task planner,
- mutex constraints imposed by task allocation,
- and transition constraints imposed by the motion planner and that account for the time required for robots to move between assigned task.

A **mutex constraint** is a relationship between two tasks, τ_i and τ_j, that ensures that either τ_j must finish before τ_i starts or τ_i must finish before τ_j starts [14]. These mutex constraints are created when a robot is assigned to a task τ_i making it unable to perform another task τ_j at the same time as τ_i.

To find a schedule that satisfies these constraints, the scheduler uses a three-part scheduling approach. The first part converts the plan into a Simple Temporal Network (STN) [33], which provides a graphical representation of the start and end times for each task, where precedence constraints separate tasks. STNs are commonly used for scheduling due to their ability to be updated and checked for consistency in polynomial time [58].

Each vertex in the STN represents a time point, and each weighted directed edge represents inequality constraints between two time-points. When converting the plan to the STN, two vertices are created for each task, with one representing the start of the task and the other representing the end of the task. For each task, an edge is added from the vertex representing its start time point to the vertex representing its end time point with the duration of the task as its weight. Additionally, each precedence constraint, $\tau_i \prec \tau_j$, is added as an edge between the vertex representing the end time point of τ_i to the vertex representing the start time point of τ_j.

To compute a minimum makespan schedule for an STN, we use a variant of the single-source shortest path algorithm [33]. The schedule calculated at this point does not include the duration of any motion plans, nor does this schedule consider the mutex constraints imposed by the allocation. We use this schedule as the σ_{LB} used by NSQ. This is displayed as σ_{LB} in Figure 4.5.

The second part of this approach adds in the mutex constraints from the allocation. This

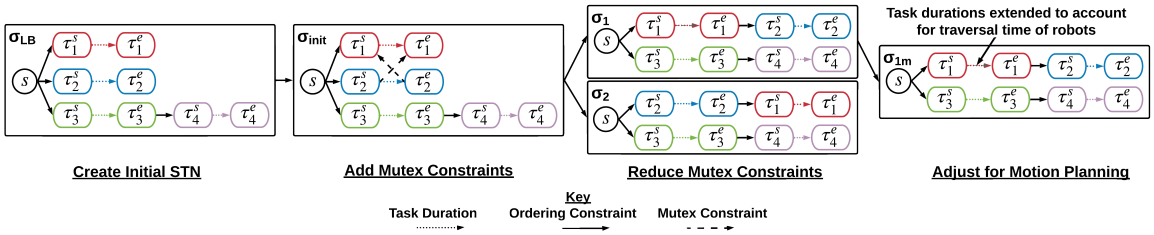


Figure 4.5: An example of the scheduling layer's process. Within each box solid lines represent precedence constraints, dashed lines represent mutex constraints, and dotted lines represent the duration of a task.

can be seen in σ_{init} in Figure 4.5. Because the task allocation layer assigned the paramedic robot to task 1 and 2, these two tasks cannot happen simultaneously; however, there is no constraint that orders one with respect to the other. The scheduler needs to convert mutex constraints into precedence constraints by selecting an ordering. A mutex constraint between two tasks τ_i and τ_j can be converted to a precedence constraint by selecting that either τ_i must finish before τ_j starts ($\tau_i \prec \tau_j$) or vise versa. The two options for converting the mutex constraints in σ_{init} to precedence constraints are shown as σ_1 and σ_2 in Figure 4.5. To choose which set of precedence constraints the mutex constraints should be converted to, the scheduler performs a Tabu search [59] over the possible orderings while minimizing the schedule's makespan. This Tabu search does a local search over a space of possible orderings and allows us to find a conversion of mutex constraints that reduces the overall makespan of the plan.

In the third part of this approach, the scheduler adjusts the schedule to account for the execution of motion plans through the addition of transition constraints. This is represented by σ_{1m} in Figure 4.5. In this step, the motion planner determines whether every required motion plan is feasible. If feasible, a set of motion plans $\grave{X}$ is returned to the scheduling layer. The scheduler uses the length of each motion plan to determine the execution duration of the motion plan. The execution duration for each motion plan is then added to the schedule as a transition constraint. These transition constraints ensure that there is sufficient time for inter-task travel for every robot in a coalition. Furthermore, if a task

requires a robot or coalition to move then the time needed to move must be added to the static duration of the task.

Using this three-part approach, the scheduler seeks to find a valid schedule for each plan and allocation. If the scheduler cannot find a feasible schedule for a given plan and allocation, the task allocation layer is alerted. If alerted, the allocation layer prunes this allocation from its search tree. A schedule can be infeasible if:

- the STN for σ_{LB} is temporally inconsistent,
- no temporally consistent STN can be found during the Tabu search,
- no motion plan can be found for one of the queries (if infeasibility can be determined)
- no motion plan can be found within a user-defined timeout for one of the queries,
- or the addition of the transition constraints causes the STN to become temporally inconsistent.

Constraints generated by the motion planning layer will be discussed down below, however, if the STN for σ_{LB} is temporally inconsistent then the most current partial-order plan is pruned from the task planning search because the task would have to occur before itself and if no temporally consistent STN can be found during the tabu search then the allocation node is pruned from the tree.

If the scheduling layer can find a feasible schedule $\grave{\sigma}$, the makespan of this schedule $C_{\grave{\sigma}}$ is returned to the task allocation layer and is used in NSQ.

Finally, the scheduler also computes the makespan of σ_{UB}. For computational efficiency, we approximate an over-estimation of the makespan of the worst possible schedule without having any robots slow down or wait,

$$C_{\sigma_{UB}} = \frac{2Mz}{w} + \sum_{m=1}^{|\pi|} dur(\tau_m)$$

in which $|\pi|$ is the number of tasks in π, $dur(\tau_m)$ is the static duration of a task τ_m, z is the length of the longest possible path in $\mathcal{C}$, and w is the speed of the slowest robot. The

longest possible length path was calculated as the sum of the perimeter of every obstacle in the environment as well as the perimeter of the environment's bounding box.

4.5.5 Motion Planning

The motion planning layer uses the world W and the set of initial and terminal configuration spaces for all tasks in $\mathcal{T}$, $\mathcal{C}_{\mathcal{T}}$, to compute a path between two configurations for a robot or coalition while respecting obstacles that the robot or coalition must avoid. The specific algorithm used to compute a path is generic and can be substituted with any suitable motion planner. This layer memoizes the path for each specific pair of configurations and specific robot or coalition to reduce the computational cost of generating motion plans, which the motion planning layer can then use in subsequent queries. Additionally, by only computing the paths when queried by the scheduling layer, the motion planning layer performs a lazy evaluation and further reduces the computation cost. As mentioned in the previous section, the memoization creates a positive constraint containing both the query with the two configurations and the motion plan found. The motion plan can be utilized by other robots of the same species.

If the motion planning algorithm can determine infeasibility or the motion planning algorithm cannot find a motion plan within a user-defined timeout, then the motion planning layer generates a negative constraint. If the motion planning query that failed was for the initial transition between a robot r_i's initial configuration and its first task τ_j, then this solely creates the constraint that robot r_i cannot participate in task τ_j. During the first expansion of the root node, we can quickly check which specific robots can reach the initial configuration for a task and determine which robots cannot participate in each task. This reduces the branching factor for the remainder of the search and increases efficiency. If the motion planning query was for part of the execution of a task τ_j, then a constraints can created that states that robot r_i's species cannot participate in task τ_j. If the motion planning query was for a transition between tasks τ_j and τ_k, then the constraint created

states that robot r_i's species cannot participate in both tasks τ_j and τ_k, however the are no constraints place on the robot r_i's species participating in each task individually. This is because its possible the robot's ability to reach either of τ_j or τ_k depends on its previous location (e.g., the robot could be good at pushing doors open, but struggle to pull doors open making motion plans irreversible).

Due to the fact the motion planning computes each motion plan independent of the other motion plans, this layer cannot compute robot-on-robot collisions. Additionally, objects in the world change location as robots manipulate them. As such, a robot's free configuration space while executing a task τ_i may depend on the order in which tasks are executed. We assume that modern motion controllers used by the robots during execution will prevent robots from colliding with one another and with objects that have been moved in most domains [82, 87, 100]. In future work, we hope to improve this portion of the framework to consider these types of constraints.

4.6 Experimental Evaluations

We evaluated GRSTAPS through five experiments in a simulated emergency response domain widely used in other works [19, 61, 62, 63]. In this domain, a heterogeneous team of robots must work together to rescue wounded survivors, deliver medicine and supplies to hospitals, put out fires, and repair damaged structures. Achieving each of the above goals requires the execution of several tasks. For example, to effectively rescue a survivor in this domain, an appropriate subset of robots need to travel to each survivor, carry them to a hospital, and deliver medicine to the hospital to heal the survivor. We generate different simulated problem instances by varying the number of survivors, fires, and buildings.

We compared the performance of GRSTAPS against two types of baselines. First, against two sequential versions of GRSTAPS that do not interleave information across computational layers (Section 4.6.2). Second, against three state-of-the-art temporal planners (Section 4.6.3). In addition to our comparative experiments, we also tested the limits of

GRSTAPS in terms of its ability to scale with the number of robots (Section 4.6.4).

In all experiments, we used maps from the Robocup Rescue Competition [60]. Each map contained buildings, composed of various polygonal shapes, and roads. An example map is shown in Figure 4.6. For each problem instance, we randomly assigned a hospital, fire station, and construction company to buildings. Similarly, survivors, fires, and rubble were randomly assigned locations on the map. Note that these locations could overlap (e.g., a survivor can be in a building that needs to be repaired).

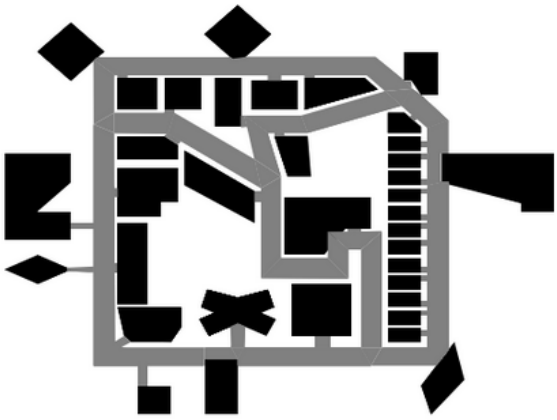


Figure 4.6: Example survivor domain map used for the experiments.

For all experiments, we used a Lazy PRM [64] motion planning implementation from the Open Motion Planning Library (OMPL) [65] as part of the motion planning layer for GRSTAPS. For our implementation of GRSTAPS, a configuration represents a 2-dimensional pose of a robot and does not consider the dynamics of the robot. However, it should be noted that higher dimensional configurations and dynamics can be used within the theoretical framework with a different choice of motion planning implementation.

We ran all experiments on an i7-8565 CPU with 8GB of RAM. The complete source code for GRSTAPS and scripts used for experiments can be found at https://github.com/amessing/grstaps.git.

4.6.1 Metrics

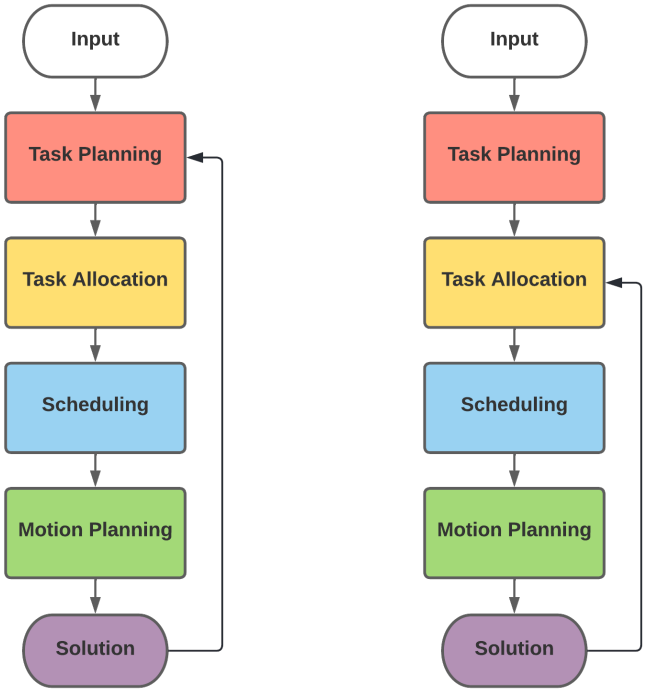
We evaluated the performance of GRSTAPS and the various baselines using the following metrics:

- *Computation Time*: the total time needed for an algorithm to solve a specific STAP-STC problem.
- *Coverage Percentage*: the percentage of the problems that a specific algorithm successfully solved.
- *Makespan*: the total time from the start of the first task in a schedule to the completion of the last task in the schedule.
- *Travel time*: the total time that each agent spends traveling between tasks that it is assigned.
- *Task Planning Nodes Expanded*: the number of nodes that are expanded during the search in the task planning layer.
- *Task Planning Nodes Visited*: the number of nodes that are visited during the search in the task planning layer.
- *Task Allocation Nodes Expanded*: the number of nodes that are expanded during the search in the task allocation layer.
- *Task Allocation Nodes Visited*: the number of nodes that are visited during the search in the task allocation layer.
- *Average Number of Concurrent Tasks*: the average number of tasks that occur simultaneously throughout the schedule.

4.6.2 Comparisons with Sequential Baselines

For the first two experiments, we compared GRSTAPS against two baselines. Both baselines are sequential anytime variants of the GRSTAPS framework and do not interleave information between layers. As such, the two baselines simulate chaining individual solu-

tions to each of the four separate sub-problems. Note that the sequential operation of the baselines differs significantly from GRSTAPS' recursive hierarchical approach that interleaves information from the lower layers to guide the search in higher layers. We designed these comparisons to evaluate the benefits of interleaving modules within GRSTAPS. We call these two anytime algorithms Sequential Task Planning Anytime (STPA) and Sequential Task Allocation Anytime (STAA).



(a) STPA (b) STAA

Figure 4.7: High-level architecture of STPA/STAA.

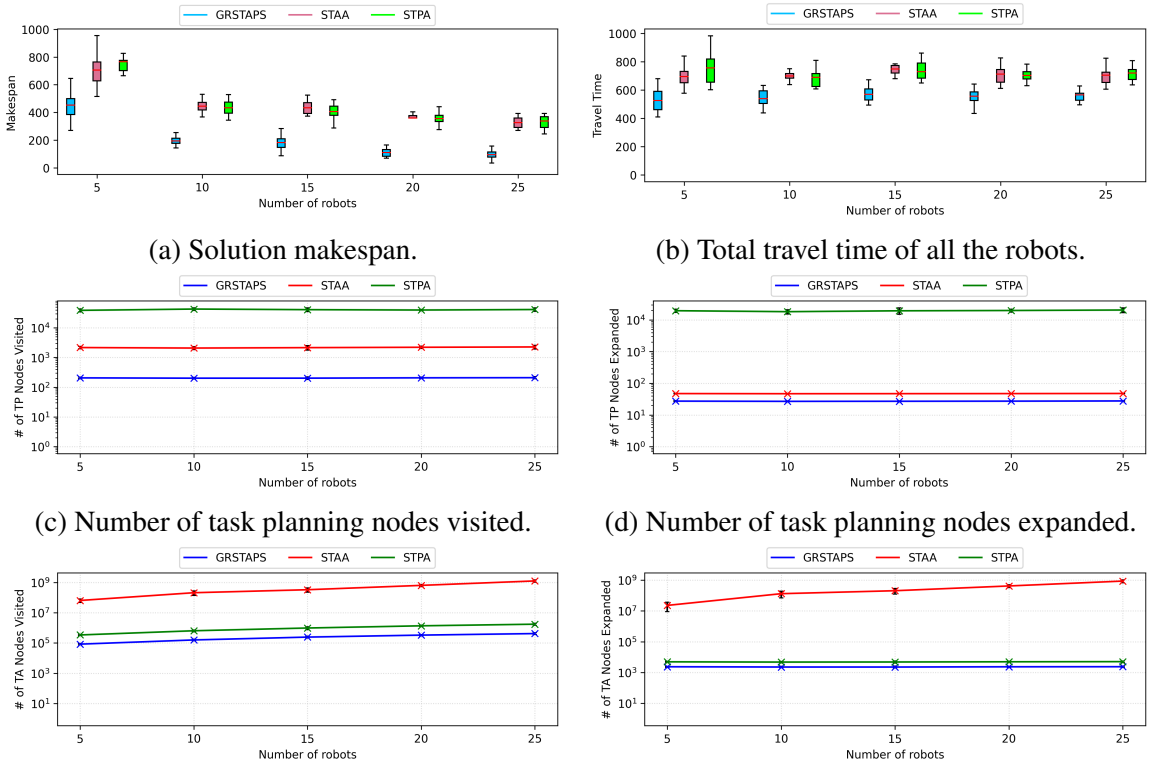
As anytime algorithms, STPA and STAA first find an initial solution to the STAP-STC problem and then iteratively refine that solution. Both STPA and STAA use the same process to find an initial solution: they search the plan space for $\hat{\pi}$. Instead of using the information from the shared constraints to prune the plan space, reduce the branching factor, and to compute costs, STPA and STAA solely use information from the task planning layer to construct $\hat{\pi}$. Upon finding $\hat{\pi}$, the task planning layer creates a desired trait matrix $\mathbf{Y}^{\hat{\pi}}$. After receiving $\hat{\pi}$ and $\mathbf{Y}^{\hat{\pi}}$, task allocation searches the incremental task allocation space for

$\hat{\mathbf{A}}$ using only APR and without considering any associated schedules. Upon finding $\hat{\mathbf{A}}$, $\hat{\mathbf{A}}$ and $\hat{\pi}$ are passed to scheduling. Scheduling then builds the schedule using the same process as GRSTAPS. When finding the initial solution, both baselines move back up a layer only if they cannot find the solution after searching the entire space for the current layer. Finally, motion planning is called to calculate the motion plans required by the scheduling layer. For example, after finding the first $\hat{\pi}$ the baselines will not come back to the task planning layer unless the entire incremental task allocation space is searched and no solution is found. When compared to GRSTAPS, these two baselines give an understanding of how the interleaving of GRSTAPS through shared constraints improves performance.

The difference between these two baselines arises in how they refine after finding an initial solution (see Figure 4.7). After finding an initial solution, STPA continues to use FCPOP in an attempt to find another $\hat{\pi}$ that has a lower associated makespan. Upon finding another $\hat{\pi}$, it then passes this plan through the rest of the process is used to find an initial solution, including task allocation, scheduling, and motion planning. If it finds another solution, it returns to the task planning layer and continues the search. This refinement process continues until a timeout occurs.

In contrast, after finding an initial solution, STAA continues the search in the incremental task allocation space. It attempts to find another allocation $\hat{\mathbf{A}}$ with a lower associated makespan. It only returns to the task planning layer upon searching the entire incremental task allocation space and similarly runs until a timeout occurs.

We ran all three algorithms – GRSTAPS, STPA, and STAA – for the same amount of time in order to provide each algorithm an equal opportunity to find a good solution. The time was selected as the amount of time that it took GRSTAPS to find a solution. This time was always greater than the time needed by STPA and STAA to identify an initial solution, meaning both baselines had time to refine their solutions.



(a) Solution makespan.

(b) Total travel time of all the robots.

(c) Number of task planning nodes visited.

(d) Number of task planning nodes expanded.

(e) Number of task allocation nodes visited.

(f) Number of task allocation nodes expanded.

Figure 4.8: Experiment 1: A comparison of GRSTAPS to both sequential baselines (STPA and STAA) when scaling the number of robots. All problems have 20 goals.

Scalability with increasing team size against sequential approaches

The first experiment tested the ability of GRSTAPS to scale with an increasing number of robots in comparison to the sequential baselines. For this experiment, we tested five different team sizes: 5, 10, 15, 20, and 25 robots. For each team size, we randomly generated 20 problem instances. All problem instances had 20 goals (10 survivors, 5 fires, and 5 damaged buildings to repair). The results of these experiments are summarized in Figure 4.8.

As shown in Figure 4.8a, GRSTAPS created solutions with lower makespans than both of the sequential baselines. On average, the makespan of STPA and STAA solutions was 268% and 274% larger than those of GRSTAPS, respectively.

While each approach searched the same plan space for all 100 problems, the infor-

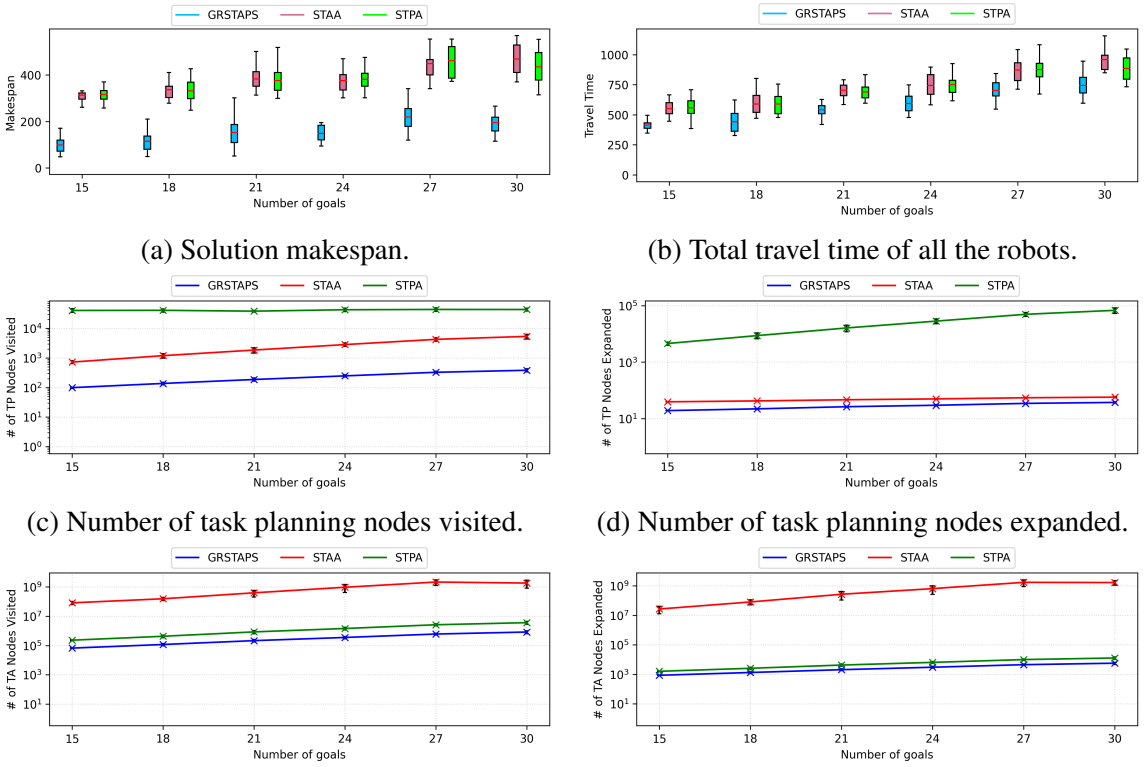
mation utilized by each approach to search the plan space greatly contributes to why GRSTAPS created solutions with lower makespans. STPA and STAA only consider the agent-agnostic tasks when building the task plan, so they do not consider the proximity of tasks to one another or any constraints placed upon the ordering of tasks by the assignment of robots and their tasks proximities. GRSTAPS, on the other hand, evaluates plan nodes based on a schedule that includes the travel times of robots. This implicitly considers the proximity of tasks to each other and the proximity of the robots to their assigned tasks. GRSTAPS is thus able to conduct a more informed search through the plan space and select a task plan that minimizes the overall makespan when the other constraints are considered.

At the allocation level, STPA and STAA conduct a search through the incremental task allocation space without considering the resulting schedule. As both STPA and STAA use APR to guide their search, they aim to find an allocation that satisfies $\mathbf{Y}^{\hat{\pi}}$ by assigning as few robots as possible. This causes them to prefer robots with larger trait values regardless of the robots' other assignments. As a result, robots with larger trait values get assigned multiple tasks while other robots remain under-utilized. As no robot can perform more than one task at any given time, this reduces concurrency and can lead to a longer makespan. Additionally, when STPA and STAA assign robots to tasks, they do not consider the robot's proximity to the task. This can lead to a robot needing to travel a far distance to reach its assigned task, which delays the start of that task and leads to a longer makespan. As GRSTAPS considers both the schedule and the motion plans, it can conduct a more informed search through the incremental task allocation space and selects an allocation that minimizes makespan. In Figure 4.8b, one can see that the solutions generated by GRSTAPS required the robots to travel less to reach each assigned task than both STPA and STAA.

Figures 4.8c and 4.8d show the number of task planning nodes visited and expanded. As all three approaches use agent-agnostic tasks, the change in the number of robots had a smaller impact on the task planning search, which resulted in a small variation in the num-

ber of task planning nodes visited and expanded. As can be seen, GRSTAPS both visited and expanded fewer task planning nodes than STPA and STAA. This is likely due to its more informed search and because it prunes part of the plan space based on the shared constraints from the task allocation, scheduling, and motion planning layers. Because STPA and STAA do not use this interleaved process, they can find a task plan $\hat{\pi}$ that satisfies G, but which cannot be allocated or one for which motion plans cannot be created. Upon finding such a task plan, they start searching locally in the plan space as the nodes close to $\hat{\pi}$ have better heuristic values. This can cause them to get stuck in the local minimum of plan space surrounding the $\hat{\pi}$. Furthermore, STPA runs an anytime approach at the task planning level, so after it finds an initial solution, it continues to search the plan space for a task plan that is part of a better solution. This causes it to visit and expand significantly more task planning nodes than either GRSTAPS or STAA.

Figures 4.8e and 4.8f show the number of task allocation nodes that each approach visited and expanded. As the number of robots increases, the number of task allocation nodes visited and expanded increases exponentially. However, GRSTAPS both visited and expanded fewer task allocation nodes than STPA and STAA. Similar to the search through the plan space, GRSTAPS conducts a more informed search of the incremental task allocation space and uses information from the shared constraints to prune portions of this space. Also, since these baselines allocate without considering the schedule or required motion plans, they can find an $\hat{\mathbf{A}}$ that cannot be scheduled or one for which motion plans cannot be created. Upon finding such an allocation, they will start searching locally in the incremental task allocation space as the nodes close to $\hat{\mathbf{A}}$ will have better heuristic values. This can cause them to get stuck in the local minimum of incremental task allocation space surrounding the $\hat{\mathbf{A}}$. Furthermore, STAA runs an anytime approach at the task allocation level, so after it finds an initial solution, it continues to search the incremental task allocation space for an allocation that is part of a better solution. This causes it to visit and expand significantly more task allocation nodes than either GRSTAPS or STPA.



(a) Solution makespan.

(b) Total travel time of all the robots.

(c) Number of task planning nodes visited.

(d) Number of task planning nodes expanded.

(e) Number of task allocation nodes visited.

(f) Number of task allocation nodes expanded.

Figure 4.9: Experiment 2: A comparison of GRSTAPS to both sequential baselines (STPA and STAA) when scaling the number of goals. All problems have 15 agents.

Scalability with increasing number of goals against sequential approaches

The second experiment tested the ability of GRSTAPS to scale with an increasing number of goal conditions compared to the sequential baselines. This comparison allowed us to gain insights into how the GRSTAPS scales as a function of the planning problem's size. For this experiment, the number of goals was varied from 15 to 30 in increments of 5. We ran 20 problem instances for each set of goals. All problem instances used 15 robots. The results of this experiment can be seen in Figure 4.9.

As shown in Figure 4.9a, when compared to STPA and STAA, GRSTAPS creates solutions with lower makespans than both of the sequential baselines. On average, STPA generated a solution with a makespan that was 285% of the one GRSTAPS generated. STAA generated a solution with a makespan that was 289% of the one GRSTAPS gener-

ated. This is caused by the same reasons as in Section 4.6.2. Once again, GRSTAPS uses a more informed search through both task planning and task allocation spaces. This more informed search leads to less total travel time by the robots as seen in Figure 4.9b and a lower overall makespan.

Figures 4.9c and 4.9d show the number of task planning nodes visited and expanded. As the number of goals increases, the overall plan space and the length of the task plan needed to achieve these goals become longer. This leads to an increase in the number of task planning nodes visited and explored. As can be seen, GRSTAPS visits and explores fewer task planning nodes than either STPA and STAA.

Figures 4.9e and 4.9f show the number of task allocation nodes that each approach visited and expanded. As mentioned in the previous paragraph, as the number of goals increases, the number of task planning nodes visited increases. This propagates and increases the number of task allocation nodes visited and expanded. As can be seen, GRSTAPS visits and explores fewer task allocation nodes than either STPA and STAA.

4.6.3 Comparisons with Temporal Planner Baselines

For the third and fourth experiments, we compared GRSTAPS against three state-of-the-art temporal task planners: FCPOP [63], OPTIC [16, 115], and TFLAP [15, 39]. These planners have been shown to be highly effective and efficient at temporal planning when compared to prior state-of-the-art temporal planners. TFLAP was the highest performing non-portfolio planner at the 2018 International Planning Competition's temporal track [20]. OPTIC was the baseline used for this competition. FCPOP is from our prior work and has been demonstrated to be competitive against these other temporal planners.

However, none of the temporal planners can compute motion plans or allocate coalitions based on traits. To resolve this and ensure fair comparison, we used a preprocessing step to formulate the problem in a way that was compatible with their planning formulation. For each problem, the preprocessing stage first generated a fully connected graph

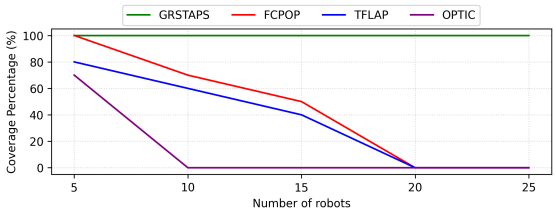
where each node is a location (e.g., the hospital, survivor 1's initial location, etc.), and each edge is labeled with the traversal time for each robot/coalition. In order to calculate the traversal times, the algorithm queried the motion planning module from GRSTAPS for motion plans between each pair of nodes. The length of each motion plan and the speed of each robot/coalition was then used to compute the traversal times for each edge.

Next, the preprocessing stage accommodated for the fact that these temporal planners cannot handle trait-based models. We converted each agent-agnostic task into a set of non-agent-agnostic tasks. These tasks each consider which robots were assigned to execute them. This was done by computing a set of all possible coalitions. For each agent-agnostic task τ, the preprocessing step checked each possible coalition $coal$ to determine if the coalition's combined traits satisfied the task traits requirements vector y^{τ} for τ. If the coalition's combined traits satisfied y^{τ}, then a non-agent-agnostic task τ^{coal} was created, and the duration of the non-agent-agnostic task was updated based on the time the coalition needed to complete it. Additionally, move tasks were added for each individual robot to travel from one location to another. The temporal planners used the non-agent-agnostic and move tasks to solve the problem.

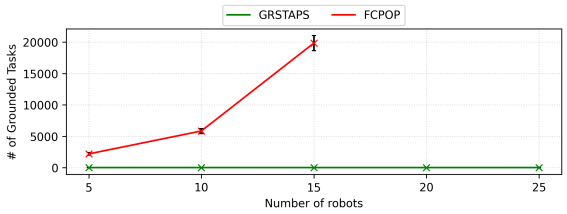
By adding these tasks via the preprocessing steps, the temporal planners are effectively searching through a discretized version of the entire problem space.

Scalability with increasing team size against temporal planners

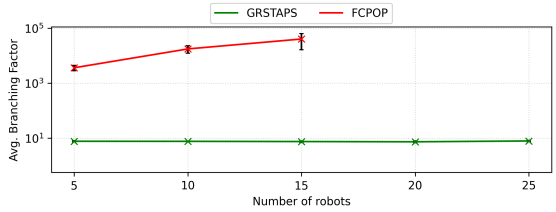
For this experiment, we used five different team sizes. For each team size, we randomly generated 20 problem instances. All problem instances had 15 goals (7 survivors, 4 fires, and 4 damaged buildings to repair). For each problem instance, we ran each algorithm until it either solved the problem instance, ran out of memory, or exceeded a 10-minute timeout. We reduced the number of goals from experiment 1 due to the time required for each of the temporal planners to solve larger problem instances. Since we could not measure the number of grounded tasks and average branching factor for TFLAP and OPTIC without



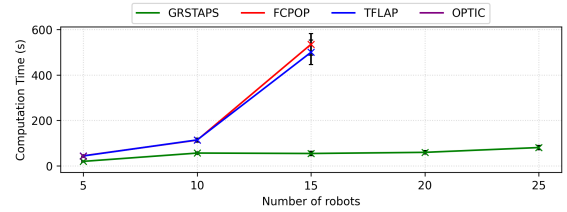
(a) Comparison of GRSTAPS to FCPOP, TFLAP, and OPTIC in terms of problem coverage.



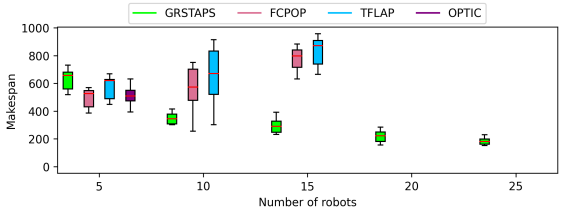
(b) Comparison of GRSTAPS to FCPOP, TFLAP, and OPTIC in terms of total number of grounded tasks.



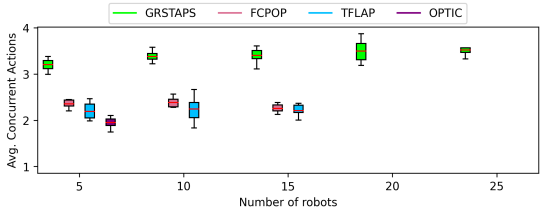
(c) Comparison of GRSTAPS to FCPOP, TFLAP, and OPTIC in terms of average branching factor of planning search.



(d) Comparison of GRSTAPS to FCPOP, TFLAP, and OPTIC in terms of computation time.



(e) Comparison of GRSTAPS to FCPOP, TFLAP, and OPTIC in terms of solution makespan.



(f) Comparison of GRSTAPS to FCPOP, TFLAP, and OPTIC in terms of the average number of tasks that happen concurrently.

Figure 4.10: Experiment 3: A comparison of GRSTAPS to three temporal planners (FCPOP, TFLAP, and OPTIC) when scaling the number of robots. All problems have 15 goals.

modifying their source code, we only report those metrics for FCPOP in Figures 4.10b and 4.10c. We believe that TFLAP and OPTIC would have generated the same number of grounded tasks as the total number of grounded tasks is problem instance-specific.

Figure 4.10a shows each approach's problem coverage for each team size. As can be seen, as the number of robots increased, FCPOP, OPTIC, and TFLAP struggled to find solutions to the planning problems. While GRSTAPS solved all 100 problems, FCPOP, TFLAP, and OPTIC solved only 44, 36, and 14 problems, respectively. In fact, OPTIC could not solve any problem with more than 5 robots, and FCPOP and TFLAP could not

solve any with more than 15 robots. This is likely due to the fact that, as the number of agents increases, for the temporal planners, the number of grounded tasks (allocated tasks and move tasks) increases combinatorically (see Figure 4.10b). As the team size increases, GRSTAPS maintains the same number of grounded tasks due to agent-agnostic planning. Additionally, the temporal planners have a grounded task for each possible coalition, task schema, and parameter combination as they are searching the entire problem space on one layer. The larger number of grounded tasks also leads to a much higher branching factor. As shown in Figure 4.10c, GRSTAPS has a fairly consistent average branching factor even when the team size increases. However, the temporal planners have a larger average branching factor, and the average branching factor grows as the team size grows.

The larger number of grounded tasks and higher average branching factor increased the computation time for the temporal planners. However, as can be seen in Figure 4.10d, even for the problems that the temporal planners can solve, GRSTAPS solved them faster. While all of the approaches took longer to solve the problem as the team size increased, the increase in team size had a much more substantial impact on the temporal planners' computation times.

Finally, we measured the makespan of the solution for each problem. As can be seen in Figure 4.10e, FCPOP, OPTIC, and TFLAP generated better solutions for teams of 5 robots. However, they created worse solutions (i.e., longer makespan) or no solutions for larger team sizes. Part of the reason GRSTAPS creates plans with lower makespans is that it creates plans with higher concurrency, as seen in Figure 4.10f. As a result, GRSTAPS' solutions accomplish more simultaneously and thus are more compact. The increase in average branching factor and number of grounded tasks are also contributing factors to the temporal planners' lower concurrency. Each of their searches implicitly (and explicitly in the case of TFLAP) minimizes the number of tasks in the plan in addition to trying to minimize the overall makespan.

Scalability with increasing number of goals against temporal planners

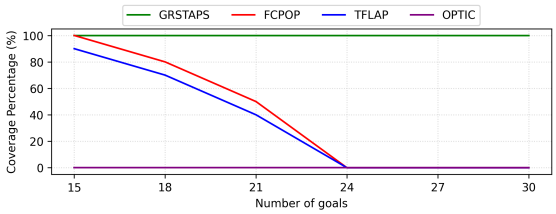
For the fourth experiment, we varied the number of goals from 15 to 30. For each number of goals, we randomly generated 20 problem instances. All problem instances had 15 robots each. For each problem instance, we ran each algorithm until it either solved the problem instance, ran out of memory, or exceeded a 10-minute timeout. OPTIC was unable to solve any of the problem instances and so is excluded from the results below.

Similarly to experiment 3, measuring the number of grounded tasks and average branching factor for TFLAP would require changing its code and so it is excluded from Figures 4.11b and 4.11c. Also, similarly to experiment 3, TFLAP should have the same number of grounded tasks as FCPOP as the total number of grounded tasks is problem instance-specific.

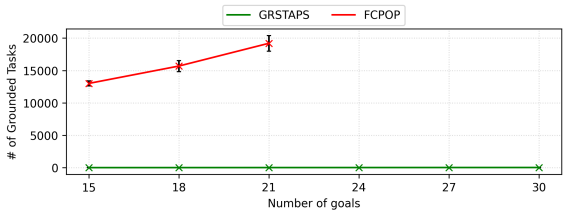
For this experiment, FCPOP and TFLAP were able to solve some of the problem instances up to 21 goals, while OPTIC was unable to solve any of the problems. As the number of goals increased, the percentage of problems that FCPOP and TFLAP could solve decreased. In total GRSTAPS solved all 120 problems, while FCPOP and TFLAP solved 46 and 40 respectively (see Figure 4.11a). This is largely due to the significantly higher number of grounded tasks (see Figure 4.11b) and the higher average branching factor (see Figure 4.11c).

While GRSTAPS has a more computationally expensive heuristic at the task planning layer due to its task allocation and motion planning layers, the significantly larger number of branches that each of the temporal planners has to consider at each iteration of the search causes them to be slower. As can be seen in Figure 4.11d, as the number of goals increased, the separation in the amount of time needed to solve a problem between GRSTAPS and each of the temporal planner increased until they cannot solve the problem within the timeout. This limits temporal planners from being able to solve a problem with more than 21 goals.

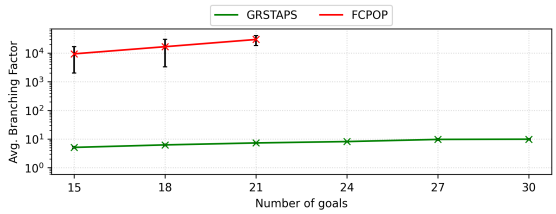
Finally, we measured the makespan of each of the generated solutions. Similar to ex-



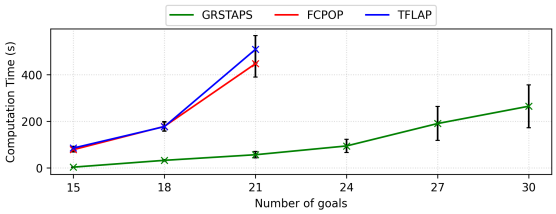
(a) Comparison of GRSTAPS to FCPOP in terms of problem coverage.



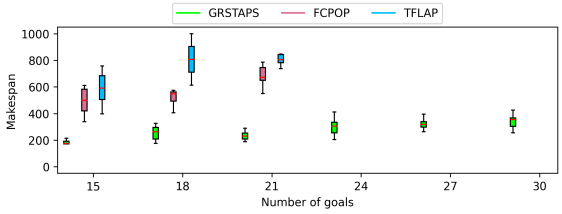
(b) Comparison of GRSTAPS to FCPOP in terms of total number of grounded tasks.



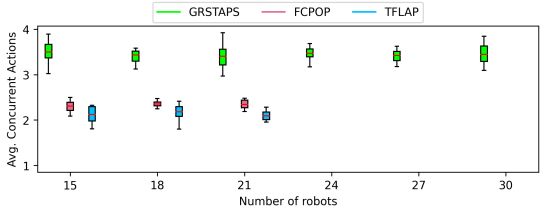
(c) Comparison of GRSTAPS to FCPOP in terms of average branching factor of planning search.



(d) Comparison of GRSTAPS to FCPOP in terms of computation time.



(e) Comparison of GRSTAPS to FCPOP in terms of solution makespan.



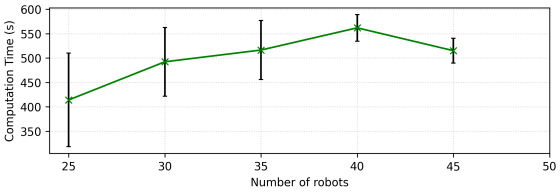
(f) Comparison of GRSTAPS to FCPOP in terms of the average number of tasks that happen concurrently.

Figure 4.11: Experiment 4: A comparison of GRSTAPS to three temporal planners (FCPOP, TFLAP, and OPTIC) when scaling the number of goal. All problems have 15 agents.

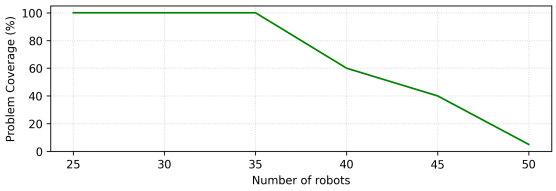
periment 3, GRSTAPS creates solutions with better makespan than the temporal planners (see Figure 4.11e). This is partially due to GRSTAPS' ability to create compact plans with higher average concurrency (see Figure 4.11f).

4.6.4 Testing the limits

In the final experiment, we tested the limits of GRSTAPS in terms of its scalability with team size when tasked with 30 goals. For each team size, there were ten problem instances randomly generated. For each problem instance, GRSTAPS was run until it either solved the problem instance or exceeded a 10-minute timeout. We continued increasing the team



(a) Computation time results for testing the limits of GRSTAPS with 30 goals and an increasing number of agents.



(b) Coverage results for testing the limits of GRSTAPS with 30 goals and an increasing number of agents.

Figure 4.12: Experiment 5: Testing the limits of GRSTAPS using 30 goals and an increasing number of agents.

size until GRSTAPS was incapable of solving any of the ten problems within the 10-minute limit. The results from these experiments can be seen in Figure 4.12.

As shown in Figure 4.12a, with 30 goals, GRSTAPS solved STAP-STC problems with up to 45 agents within the 10-minute limit. However, for team sizes of 40 and 45, it was not able to solve all of the problem instances.

The increased number of goals compared to prior experiments required GRSTAPS to handle larger task planning spaces and longer plans. This increases both the number of times the task planning layer calls the task allocation layer and the size of the task allocation space for each call. Additionally, as the number of robots increases, the size of the task allocation space further increases. This leads to more time spent to allocate the tasks. In turn, the increase in the number of allocations leads to more calls to the motion planner. While the memoization and lazy-evaluation mechanisms in the motion planning layer improves computational efficiency, creating a new motion plan is time-consuming. All of these factors together influence the largest problem size that GRSTAPS can handle when operating under a certain time limit.

4.7 Discussion & Conclusion

In this chapter, we have formalized a new class of heterogeneous multi-robot problems named *Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-*

STC). To our knowledge, our formalization represents the first attempt to fully integrate four interconnected problems: task planning, allocation, scheduling, and motion planning. In addition, we introduced an interleaved framework named *Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling (GRSTAPS)* that performs a multi-layer search to solve STAP-STC problems.

Through detailed ablation studies, we were able to show that the interleaved approach used by GRSTAPS delivers improved performance in terms of solution quality over two sequential anytime baselines that do not interleave the execution of the individual modules; in particular GRSTAPS was able to find plans with lower makespan more quickly than all baselines.We also demonstrated that GRSTAPS consistently outperforms state-of-the-art temporal planners both in terms of solution quality and efficiency. Finally, we demonstrated that GRSTAPS can solve STAP-STC problems with up to 30 goals and 45 robots within 10 minutes.

However, GRSTAPS does have its limitations and there are open questions left to explore. First, as GRSTAPS builds on top of ITAGS it shares the limitations about completeness, assuming static traits, and a deterministic fully-observable problem as discussed in Chapter 3.

Additionally, the current implementation utilizes some simple motion planning algorithms that while fast cannot consider collisions between robots. For the examples considered in this paper, tight coupling of robot motion was not required (there were no instances of cooperative manipulation, and no narrow passageways that could lead to deadlock), so this limitation has not been problematic. For applications in which more intricate robot-to-robot interaction is required (e.g., multiple mobile manipulators cooperating to perform household tasks), it will be essential to include robot interaction when planning collision-free paths. With regard to mobility, more advanced motion planners, such as those in [25, 116], may be more suitable for teams of heterogeneous robots, while for shared manipulation, some combination of sampling-based planning in composite configuration spaces and

low-level cooperative control (during execution) may prove effective. Another limitation is the interface between symbolic tasks and their geometric counterparts.

There is a significant amount of research from the Task and Motion Planning community that focuses on the interaction between symbolic task descriptions and their corresponding geometric motions, and in particular to the potential for unanticipated conflict at the geometric level once a symbolic task plan has been determined. In our current implementation, we have addressed these difficulties by making two simplifying assumptions. First, we rely on a simplistic static representation of the environment that does not update robots' free configuration spaces when robots rearrange objects in the work space. Again, because the examples considered in this paper did not require close interaction between robots and objects in the environment, this limitation was not problematic. Second, at the motion planning level, we ignore geometric details for goal positions of objects that are manipulated.For example, when a robot moves a patient to the hospital, we consider the geometry of the transport path, but we do not explicitly consider the geometric description of the patient's goal location (e.g., we leave the details of placing the patient on a particular bed to the online controller). Going forward, there are opportunities to eliminate both of these simplifications, incorporating ideas from the TAMP community, such as [104, 90].

Finally, GRSTAPS considers an offline problem where it assumes a solution can be executed perfectly, however, in real problems there are lots of opportunities for execution to fail. An easy solution would be to resolve from the new initial state, however this would be time consuming when a lot of information in the previous solution, the previous search, and the previous shared constraints could still be utilized. It is possible that the solution may only need a small repair or that a new solution can be based on part of the previous solution. How to best utilize a framework such as GRSTAPS to aid in solution repair opens several questions such as how to repair various types of execution failures, how to select what parts of the solution to continue to use, and how to select what parts of the solution to repair?

CHAPTER 5
HETEROGENEOUS COALITION SCHEDULING WITH TEMPORAL UNCERTAINTY

5.1 Introduction

In this chapter, we relax the scheduling sub-problem of the Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) problem we formalized in the previous chapter[1]. We focus this chapter on exploring temporal uncertainty with the context of this scheduling sub-problem and finish by exploring it with the context of the entire STAP-STC problem.

In real-world environments, there are numerous factors that can cause both durations of tasks and durations of task transitions to be uncertain (e.g., stale map information, battery levels causing variance in individual robots' speeds, variance in individual robots' ability to execute tasks, and unexpected delays). Without reasoning about possible temporal uncertainty, these factors can compound through the entire execution of a plan and result in the entire plan taking longer to execute than expected.

Furthermore, many Multi-Robot Coordination (MRC) algorithms use an estimate of the makespan (i.e., the time needed to execute a plan) to determine what tasks should be included in the plan and what robots should participate in executing the tasks [117]. Reasoning about uncertainty and the potential risk of delays can aid these Multi-Robot Systems in making better decisions about what tasks should be executed and who should execute the tasks.

In this chapter, we introduce a novel uncertainty-aware scheduling problem, the Het-

[1]The material in this chapter is based on:
A. Messing[*], J. Banfi[*], M. Stadler, E. Stump, H. Ravichandar, N. Roy, and S. Hutchinson, "Heterogeneous Coalition Scheduling with Temporal Uncertainty," IEEE Transactions on Robotics, 2023.
[*]Co-First Authors

erogeneous Coalition Scheduling with Temporal Uncertainty (HCSTU) problem, which explicitly considers the uncertainties in the time needed to execute a task and to transition between tasks. Existing approaches that deal with uncertainty either solve a determinized version of the problem or reformulate the problem into a more general form. However, such reformulations tend to be limited to small teams (less than 5 robots) and unimodal uncertainty [118], or can often lead to overly conservative or optimistic solutions [33]. In contrast, our approach provides guarantees on risk levels and can handle larger teams (up to 15 robots) and arbitrary distributions of uncertainty.

To solve the HCSTU problem, we present a novel sampling-based risk-aware algorithm named Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee (CS-HSSRG). Unlike approaches that either do not provide guarantees on meeting risk tolerances [119] or empirically determine a risk tolerance offset to use internally to their algorithm [120, 121], our approach allows the user to specify the desired level of risk and ensures the desired level of risk is not exceeded through a theoretical guarantee.

Our approach is agnostic to the specific underlying cause of the temporal uncertainty and the specific duration distributions. Instead, we assume access to either a model of uncertainty or a generator that can generate samples of possible task and task transition durations. The CS-HSSRG algorithm uses a small number of these samples to construct a compact (and thus typically very efficient to solve) Mixed-Integer Linear Programming formulation which considers a user-defined risk threshold for the computation of the plan, which takes the form of a partial ordering of the tasks. Due to the small number of samples used, CS-HSSRG does not immediately guarantee that the tentative duration of the plan, computed by solving the MILP formulation, meets the user-defined risk threshold. Instead, it solves a Sequential Probability Ratio Test (SPRT) that checks whether the duration associated with the computed plan respects the user-defined risk threshold. If the test is positive, CS-HSSRG directly outputs the computed schedule and associated duration; otherwise, the duration is increased, and the SPRT is repeated with the same plan. This

process is repeated until a plan respecting the user-defined risk is found.

Using extensive experimental evaluations, we demonstrate that while CS-HSSRG provides a theoretical guarantee on the tolerated risk, it does not generate overly conservative solutions and is extremely efficient at computing robust plans when compared to state-of-the-art approaches. Additionally, we test CS-HSSRG as a component of our GRSTAPS framework that we described in the previous chapter and use this variant of the framework to solve a STAP-STC problem. We show that the use of CS-HSSRG for dealing with the scheduling part of the problem results in making robust decisions about what tasks to include in the plan and what robots to allocate to those tasks.

5.2 Related Work

The problem introduced in this paper can be framed as an intersection of the more general contexts of coalition scheduling and scheduling with temporal uncertainty.

5.2.1 Coalition Scheduling

A rich body of work has addressed the multi-robot scheduling problem [44] and the closely related multi-robot task allocation (MRTA) problem [42, 43]. Our work focuses on a variant of the multi-robot scheduling problem where robots can only participate in a single task at a time, but multiple robot can form coalitions and collectively executable a single task. As such, our problem falls under the single-task (ST) robots and multi-robot (MR) tasks categorization from Gerkey and Mataric's widely used taxonomy [42] and under the Synchronization and Precedence Constraints (SP) categorization from Nunes's extension to the taxonomy [44]. We focus this section on works that fall under that same banner.

Zhang and Parker [122] develop four heuristics to address the multi-robot task scheduling problem at the coalition level. These heuristics take inspiration from processor scheduling algorithms and have provable solution bounds. Bischoff *et al.* [123] build upon the MinStepSum algorithm introduced by Zhang and Parker [122], but augment the method to

directly consider precedence constraints. They then improve the schedule through a local search with a custom neighborhood operator. Neville *et al.* [5] present Normalized Schedule Quality (NSQ) as a coalition scheduling-based heuristic for their ST-MR-TA task allocation algorithm Incremental Task Allocation Graph Search (ITAGS). Matos *et al.* [124] propose an approach that utilizes Simulated Annealing to minimize the time and distance cost of executing a task set while taking into account possible pathing conflicts. While each of these approaches has several advantages when solving a deterministic scheduling problem involving heterogeneous coalitions, none of them can reason about temporal uncertainty.

5.2.2 Scheduling with Temporal Uncertainty

In robotics, scheduling with temporal uncertainty is often formulated more generally [42, 122, 44] as either an uncertainty-based variant of a Simple Temporal Problem (STP) [33] or a Resource Constrained Project Scheduling Problem (RCPSP) [118].

A Simple Temporal Problem (STP) [33] is usually represented as a directed acyclic graph where each node represents a timepoint variable and each edge represents a precedence constraint. There are two common extensions to the STP for temporal uncertainty in the Simple Temporal Problem with Uncertainty (STPU) [125] and the Probabilistic Simple Temporal Problem (PSTP) [126]. For the interested reader, a detailed description of the STP and its two uncertain variation can be found in [127].

Fang *et al.* [128] applied STPU strong controllability reductions from [129] to propose Picard, a change-constrained strong scheduler for PSTP's capable of optimizing any schedule-related objective function. The approach is evaluated only using normal distributions, but may generate overly conservative bounds for more complex distributions as they equally distribute allowable risk to create upper and lower bounds. Brooks *et al.* [130] present a novel robustness metric for temporal networks and two sampling-based approaches for approximating the robustness of a temporal network. Santana *et al.* [131]

merge STPU and PSTP to create the Probabilistic Simple Temporal Problem with Uncertainty (PSTPU) and then present the Polynomial-time Algorithm for RIsk-aware Scheduling (PARIS), a provably polynomial time algorithm for strong scheduling of PSTPU's. However, in order to solve a scheduling problem in polynomial time, PARIS requires several assumptions that limit the complexity of the problem including the assumption that each timepoint cannot be preceded by more than one contingent link. Additionally, none of these approaches can handle the disjunctions created when a robot is assigned multiple tasks and can only execute one at a time.

The Resource Constrained Project Scheduling Problem (RCPSP) is another general scheduling problem [118]. Commonly, problems involving scheduling for heterogeneous coalitions can be generalized to the RCPSP by considering each robot as a single unique resource as described in [122]. The Resource Constrained Project Scheduling Problem with Uncertainty (RCPSPU) is an extension to RCPSP that incorporates two types of uncertainty: resource uncertainty and temporal uncertainty. We focus only on the works that consider temporal uncertainty as we assume that the set of robots (i.e., our resources) is known. A detailed description of the RCPSP and its uncertain variants can be found in [132].

Lamas and Demeulemeester [119] developed a branch-and-cut method for solving the Sample Average Approximation (SAA) [133] of the RCPSPU and a new robustness measure that determines the probability that the executed schedule will be identical to the generated baseline schedule. Fu *et al.* [120] develop Benders Accelerated Cut Creation for Handling Uncertainty in Scheduling (BACCHUS), an approach based on Bender's Decomposition [134] and optimality/feasibility cuts, for generating a scheduling with an α-robust makespan. Varakantham *et al.* [121] present two SAA-based approaches for generating a scheduling with an α-robust makespan in SAA Optimization for solving RCPSP under Uncertainty (SORU) and SORU Heuristic (SORU-H). Song *et al.* [135] use Conditional Value-At-Risk (C-Var) [136] as part of a branch-and-bound framework to solve the same

problem. While these methods discussed thus far have several advantages, some common limitations are none of them provide a theoretical guarantee, none of them are demonstrated with more than 5 resources (robots), and all of them are demonstrated only with uni-modal distributions for durations. In contrast, our presented approach provides a theoretical guarantee on its risk tolerance, is demonstrated to be more efficient even when solving problems with up to 15 robots, and is agnostic to the underlying duration distribution.

5.3 Problem Description

Scheduling precedence constrained tasks that are assigned to heterogeneous coalitions of robots requires reasoning about the effects specific task orderings and the durations from executing robot motion plans have on the resulting makespan. In this section, we first present a deterministic Heterogeneous Coalition Scheduling (HCS) problem. We then define the Heterogeneous Coalition Scheduling with Temporal Uncertainty (HCSTU) problem as an extension of the HCS problem.

5.3.1 Heterogeneous Coalition Scheduling Problem

Consider a heterogeneous team of K robots that must collectively execute N tasks. Each task must be executed by one or more robots (i.e. a coalition), but each robot can only participate in a single task at a time (ST-MR-TA).

A task τ_i is defined by a duration d_i as the amount of time needed to execute τ_i for the robots that are assigned to it and an initial transition ϕ_i as the minimum amount of time needed for all of the robots assigned to τ_i to reach it from their individual initial configurations. Tasks are temporally constrained by a set of precedence constraints $\mathcal{P}$ and a set of mutex constraints $\mathcal{M}$.

A *precedence constraint ($\tau_i \prec \tau_j$)* is a temporal relationship between two tasks τ_i and τ_j that requires that the execution of τ_i concludes before τ_j starts. Each precedence constraint $\tau_i \prec \tau_j$ has a transition duration ϕ_{ij} that defines the minimum amount of time needed for

all robots assigned to both τ_i and τ_j to travel from the terminal configuration of τ_i to the initial configuration of τ_j. If there are no robots that are assigned to both τ_i and τ_j then $\phi_{ij} = 0$.

If a robot is assigned to multiple tasks, a mutex constraint is created between each pair of tasks that the robot is assigned to participate in. A *mutex constraint ($\tau_i \leftrightarrow \tau_j$)* between two tasks, τ_i and τ_j, represents the disjunction that either τ_i must conclude before τ_j starts ($\tau_i \prec \tau_j$) or τ_j must conclude before τ_i starts ($\tau_j \prec \tau_i$). If a robot is assigned to two tasks τ_i and τ_j where there already exists a precedence constraint (e.g. $\tau_i \prec \tau_j$) then the precedence constraint supersedes the mutex constraint as there is no decision to be made on the ordering of the two tasks. As such, $\mathcal{P} \cap \mathcal{M} = \emptyset$. Each mutex constraint $\tau_i \leftrightarrow \tau_j$ has a pair of transition durations ϕ_{ij} and ϕ_{ji} that define the minimum amount of time needed for all robots assigned to both τ_i and τ_j to travel from the terminal configuration of τ_i to the initial configuration of τ_j and to travel from the terminal configuration of τ_j to the initial configuration of τ_j respectively.

The Heterogeneous Coalition Scheduling (HCS) problem is defined by

- a set of task durations $\mathcal{D} = \{d_i \mid i \in \mathcal{I}\}$ where $\mathcal{I} = \{1, \ldots, N\}$,
- a set of initial task transitions $\mathcal{X}_{init} = \{\phi_i \mid i \in \mathcal{I}\}$
- a set of precedence constraints $\mathcal{P} \subseteq \mathcal{I}^2$,
- a set of mutex constraints $\mathcal{M} \subseteq \mathcal{I}^2$,
- and a set of task transition durations $\mathcal{X} = \{\phi_{ij} \mid (i,j) \in \mathcal{P}\} \cup \{\phi_{ij} \mid (i,j) \in \mathcal{M}\} \cup \{\phi_{ji} \mid (i,j) \in \mathcal{M}\}$

The goal of the HCS problem is to select a set of task orderings ρ with one for each mutex constraint that minimizes the makespan C or total time needed to execute all of the tasks while considering the task durations and task transition durations.

We provide a formal definition of this problem as the following Mixed-Integer Linear

Programming (MILP) model:

$$min \;\; C \tag{5.1a}$$

$$s.t. \;\; C \geq c_i \qquad \forall i \in \mathcal{I} \tag{5.1b}$$

$$s_j \geq c_i + \phi_{ij} \qquad \forall (i,j) \in \mathcal{P} \tag{5.1c}$$

$$s_j \geq c_i + \phi_{ij} - M(1 - \delta_{ij}) \qquad \forall (i,j) \in \mathcal{M} \tag{5.1d}$$

$$s_i \geq c_j + \phi_{ji} - M\delta_{ij} \qquad \forall (i,j) \in \mathcal{M} \tag{5.1e}$$

$$s_i \geq \phi_i \qquad \forall i \in \mathcal{I} \tag{5.1f}$$

$$\delta_{ij} \in \{0,1\} \qquad \forall (i,j) \in \mathcal{M} \tag{5.1g}$$

where s_i is the start time for the i^{th} task, c_i is the completion time for the i^{th} task ($c_i = s_i + d_i$), δ_{ij} is a boolean indicator that is 1 when the mutex constraint between the i^{th} and j^{th} tasks ($\tau_i \leftrightarrow \tau_j$) has been reduced to the precedence constraint from the i^{th} task to the j^{th} task ($\tau_i \prec \tau_j$) and 0 when the mutex constraint has been reduced to the precedence constraint from the j^{th} task to the i^{th} task ($\tau_j \prec \tau_i$).

In the above MILP model, the objective function 5.1a and Constraint 5.1b enforce the minimization of the makespan. Constraint 5.1c applies the precedence constraints and includes the time each of the robot assigned to tasks τ_i and τ_j need to transition from τ_i to τ_j. Constraints 5.1d and 5.1e apply the disjunction of the mutex constraints. Depending on the value of the mutex indicator variable δ_{ij} only one of the two constraints is used by the optimization for each mutex constraint. This disjunction makes this a Disjunctive Temporal Problem which is an NP-Hard problem [137]. Once each of the δ_{ij}'s is set then the problem becomes a Linear Problem and can be solved in polynomial time. Constraints 5.1f act to ensure that all robots assigned to a task can reach it from their individual initial positions before the task is scheduled to start.

The solution to the HCS problem is the set of task orderings $\rho = \{\delta_{ij} \mid (i,j) \in \mathcal{M}\}$

and the minimized makespan C. The Heterogeneous Coalition Scheduling problem is a subset of the Simultaneous Task Allocation and Planning with Spatiotemporal Constraints Problem from our prior work [117].

5.3.2 Heterogeneous Coalition Scheduling with Temporal Uncertainty Problem

Building upon the Heterogeneous Coalition Scheduling Problem, we now describe the Heterogeneous Coalition Scheduling with Temporal Uncertainty (HCSTU) Problem. Unlike the deterministic version of the problem, each of the task durations d_i, task transitions ϕ_{ij}, and initial task transition ϕ_i are represented as random variables instead of constants. Furthermore, whereas the makespan can be used to evaluate the quality of a schedule in the deterministic version of the problem, when temporal uncertainty is involved, the makespan itself becomes a random variable.

Similar to existing work in more general scheduling with temporal uncertainty problems [138, 139, 120, 121], we formulate the problem as a chance-constrained optimization with the goal of finding a task ordering ρ that minimizes the α-robust makespan C_α such that

$$P(C_\rho^q > C_\alpha) \leq \alpha, \quad \forall q \in S \tag{5.2}$$

where S is the space of all possible scenarios that can sampled from the underlying stochastic scheduling problem, C_ρ^q is the minimum makespan for scenario q as determined by Equation 5.1 when the task ordering ρ is used (i.e. all δ_{ij} are constants which reduces the model to a Linear Program), and α is a risk tolerance parameter provided by the user. An individual scenario $q = \langle \mathcal{D}^q,\ \Phi_{init}^q,\ \Phi^q \rangle$ contains a set of task durations $\mathcal{D}^q = \{d_i^q \mid i \in \mathcal{I}\}$, a set of initial task transition durations $\Phi_{init}^q = \{\phi_i^q \mid i \in \mathcal{I}\}$, and a set of task transition durations $\Phi^q = \{\phi_{ij}^q \mid (i,j) \in \mathcal{P}\} \cup \{\phi_{ij}^q \mid (i,j) \in \mathcal{M}\} \cup \{\phi_{ji}^q \mid (i,j) \in \mathcal{M}\}$. Equation 5.2 states that the probability that the minimum makespan C_ρ^q for any possible scenario $q \in S$ when using the task ordering ρ is greater than the α-robust makespan C_α generated to solve

the problem is less than the risk tolerance α.

5.4 Approach

Directly solving Heterogeneous Coalition Scheduling with Temporal Uncertainty Problems without sampling would require solving a chance-constrainted problem with probabilistic task durations and probabilistic task transition durations that create joint probabilistic constraints on the start and end timepoints of tasks and joint probabilistic constraints on the solution makespan. There are two main difficulties [140, 130] that make directly solving a chance-constrained problem that contains a *joint probabilistic constraint* impractical:

1. The probabilistic constraint is hard to compute as it requires multi-dimensional integration that adjusts the domain of integration based on time intervals [140, 130]. Solving a problem with these joint probabilistic constraints quickly becomes intractable as the number of tasks and the complexity of interdependencies grow.

2. The feasible region defined by the probabilistic constraint is not convex making even checking feasibility difficult [140, 130].

For this reason, we focus our attention on sampling-based methods, where one sample represents one of the many possible "true" scenarios. We first present the Sample Average Approximation (SAA) representation of the HCSTU problem along with with its limitations, and discuss why it cannot be used as part of our holistic multi-robot coordination framework GRSTAPS for the more general STAP-STC problem. We then present our approach Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee (CSHSSRG) which builds on elements from SAA but makes improvements for both efficiency and a theoretical guarantee on the risk tolerance.

5.4.1 Sample Average Approximation

The SAA MILP representation of the HCSTU problem is as follows:

$$min \;\; C_\alpha \tag{5.3a}$$

$$s.t. \;\; C^q \geq c_i^q \qquad \forall\, i \in \mathcal{I},\; q \in \mathcal{Q} \tag{5.3b}$$

$$s_j^q \geq c_i^q + \phi_{ij}^q \qquad \forall\, (i,j) \in \mathcal{P},\; q \in \mathcal{Q} \tag{5.3c}$$

$$s_j^q \geq c_i^q + \phi_{ij}^q - M(1-\delta_{ij}) \qquad \forall\, (i,j) \in \mathcal{M},\; q \in \mathcal{Q} \tag{5.3d}$$

$$s_i^q \geq c_j^q + \phi_{ji}^q - M\delta_{ij} \qquad \forall\, (i,j) \in \mathcal{M},\; q \in \mathcal{Q} \tag{5.3e}$$

$$s_i^q \geq \phi_i^q \qquad \forall\, i \in \mathcal{I},\; q \in \mathcal{Q} \tag{5.3f}$$

$$\delta_{ij} \in \{0,1\} \qquad \forall\, (i,j) \in \mathcal{M} \tag{5.3g}$$

$$C_\alpha \geq C^q - My^q \qquad \forall\, q \in \mathcal{Q} \tag{5.3h}$$

$$\alpha Q \geq \sum_{q=1}^{Q} y^q \tag{5.3i}$$

$$y^q \in \{0,1\} \qquad \forall\, q \in \mathcal{Q} \tag{5.3j}$$

For SAA, a scenario generator creates Q scenarios by randomly sampling from the various probability distributions in the problem. These Q scenarios are used to approximate the overall probability distribution of the problem. This parameter is domain dependent: larger Q increases the quality of the solution and the likelihood that the risk tolerance is respected, however, it also adds more boolean decision variables which increases the time needed to solve the problem. As such, when selecting Q there is a trade-off between solution quality and solving efficiency.

Constraints 5.3a-5.3g look very similar to those in the Deterministic MILP (See Equation 5.1), but there is a set of these equations for each of the Q scenarios where a superscript of q represents that variable for the q^{th} scenario. In addition, $\mathcal{Q} = \{1, \ldots, Q\}$ and y^q is a boolean indicator variable that denotes whether or not the q^{th} scenario is used when computing the α-robust makespan C_α. Constraints 5.3h-5.3j ensure the risk tolerance by

ignoring the scenarios that are outside the α-quantile.

An advantage of this approach is that it is not dependent on the specific distributions employed for a specific problem instance. In fact, as long as scenarios can be sampled this method is applicable. However, Constraints 5.3h-5.3j add another combinatoric factor when compared with the Deterministic MILP as each of $\binom{Q}{(1-\alpha)Q}$ combinations of scenarios have to be considered when determining what scenarios can be ignored. This makes the approach inefficient for solving complex problems when larger numbers of scenarios are needed to approximate the distributions in the problem.

Also, as mentioned in Section 5.1, when a scheduling algorithm is used as part of a higher level framework it is typically run numerous times. This means the inefficiency of using SAA would be exaggerated and significantly reduce the size of problems that GRSTAPS or similar higher level frameworks could solve. Furthermore, there is not a guarantee that just because the α-robust makespan C_α produced by SAA was respected by the sampled scenarios used in the SAA MILP model that the risk of a real scenario requiring more time to execute than C_α is less than the risk tolerance α.

5.4.2 Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee

In order to efficiently solve the Heterogeneous Coalition Scheduling with Temporal Uncertainty Problem, we present a sampling-based risk-aware approach named Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee (CS-HSSRG). This approach decomposes the SAA formulation into a heuristic for selecting representative samples and a simplified MILP formulation that we call S-SAA (See Equation 5.5). By utilizing a heuristic to efficiently select samples, we can remove one of the combinatoric factors and a set of boolean decision variables from the MILP formulation that our approach needs to solve. As a result, this simplified MILP formulation has the same complexity as the Deterministic MILP (Equation 5.1) and is more efficient to solve than the SAA formulation. After solving the simplified MILP formulation, our approach uses the Sequential Probabil-

ity Ratio Test (SPRT) to enforce a theoretical guarantee that the risk tolerance is respected and that the solution includes a truly α-robust makespan.

Algorithm 8: Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee

Input: risk tolerance α, # of scenarios to be used in S-SAA MILP (Equation 5.5) η, percentage to increment the makespan δ
Output: α-robust makespan C_α, a set of task orderings ρ

```
1  F ← {A large number of scenario samples}
   // Each scenario in F is labeled with a heuristic value
2  L ←{label(q) for q ∈ F}
3  H ← The α-quantile subset of F based on the labels from L
4  U ← {η − 1 random scenarios taken from H and the scenario in H with the
     largest label}
5  C_α, ρ ← Solve S-SAA MILP (Equation 5.5) with the scenarios in U
6  while true do
       // Run the Sequential Probability Ratio Test
7      if sprt(C_α, ρ, α) = success then
8          return C_α, ρ
9      else
10         C_α ← C_α * (1 + δ)
```

The high-level pseudocode for this approach is shown in Algorithm 8. We first generate a large number of scenarios F (Line 1). The number of scenarios in F should be greater than Q. For our experiments, we chose to make F an order of magnitude greater than Q. We then label each scenario in F with a heuristic value that is representative of the average makespan for that specific scenario across the difference possible mutex reductions (Line 2). We use a domain independent heuristic where we compute a weighted summation of the time needed to execute each of the tasks and transitions for that specific scenario as shown in Equation 5.4 below, where ψ_1-ψ_4 are weights:

$$\begin{aligned} label(q) = \psi_1 \sum_{i \in \mathcal{I}} d_i^q + \psi_2 \sum_{i \in \mathcal{I}} \phi_i^q \\ + \psi_3 \sum_{(i,j) \in \mathcal{P}} \phi_{ij} + \psi_4 \sum_{(i,j) \in \mathcal{M}} (\phi_{ij}^q + \phi_{ji}^q). \end{aligned} \tag{5.4}$$

We then use the labels to take the α-quantile subset of F which we will call H (Line 3). We select η scenarios from H (Line 4), where $\eta - 1$ of the scenarios are randomly selected and the final scenario selected is the scenario with the largest heuristic value in H. Lines 2 - 4 approximate what Constraints 5.3h-5.3j do in the Sample Average Approximation MILP. This allows us to use the η scenarios in a simplified version of the Sample Average Approximation MILP (S-SAA MILP) shown below. Unlike SAA, where increasing Q increases the number of boolean decision variables, in S-SAA η only adds more constants and linear constraints. This causes it to have less of an impact on the efficiency of solving the S-SAA formulation.

$$min \;\; C_\alpha \tag{5.5a}$$

$$s.t. \;\; C_\alpha \geq c_i^q \qquad \forall\, i \in \mathcal{I},\; q \in \mathcal{Q} \tag{5.5b}$$

$$s_j^q \geq c_i^q + \phi_{ij}^q \qquad \forall\, (i,j) \in \mathcal{P},\; q \in \mathcal{Q} \tag{5.5c}$$

$$s_j^q \geq c_i^q + \phi_{ij}^q - M(1 - \delta_{ij}) \qquad \forall\, (i,j) \in \mathcal{M},\; q \in \mathcal{Q} \tag{5.5d}$$

$$s_i^q \geq c_j^q + \phi_{ji}^q - M\delta_{ij} \qquad \forall\, (i,j) \in \mathcal{M},\; q \in \mathcal{Q} \tag{5.5e}$$

$$s_i^q \geq \phi_i^q \qquad \forall\, i \in \mathcal{I},\; q \in \mathcal{Q} \tag{5.5f}$$

$$\delta_{ij} \in \{0, 1\} \qquad \forall\, (i,j) \in \mathcal{M} \tag{5.5g}$$

The S-SAA MILP attempts to select a set of task orderings ρ that minimizes the makespan C_α for all η scenarios. At this point, there is no theoretical guarantee that the risk of a real scenario taking longer to execute than C_α is less than the risk tolerance α. As such, it is not known if the produced makespan is truely α-robust yet.

We use the Sequential Probability Ratio Test to test for a guarantee that the produced makespan is actually α-robust. If the test passes, then the makespan produced is α-robust and the solution is returned by the algorithm (Lines 7 and 8). If the test fails, then the

makespan is increased (Line 10) and the test is tried again until we have a makespan that is guaranteed to be α-robust.

5.4.3 SPRT for α-robustness guarantee

When a task ordering ρ and a makespan C_α are produced by the S-SAA, we do not immediately know whether C_α is truly α-robust. We utilize the Sequential Probability Ratio Test (SPRT) [13] to test the hypothesis that the risk of the execution of the task ordering ρ taking longer than C_α is less than or equal to α, i.e., $p(C_\rho^q > C_\alpha) \leq \alpha, \forall q \in S$. We choose to use the SPRT over other possible hypothesis testing algorithms because, as a sequential hypothesis test, the samples needed to confirm or reject the hypothesis do not need to be drawn in advance; instead, they are drawn on-demand until enough evidence to confirm or reject the hypothesis is collected. This makes the SPRT an efficient hypothesis test that only draws the minimum number of needed samples. A tutorial of the binomial variant of the SPRT can be found in the Appendix and an even more complete explanation of the SPRT can be found in the original paper by Wald [13].

For our approach, the SPRT is given a risk tolerance parameter α, a reference makespan C_α that we are testing, and a set of task orderings ρ. We want to decide whether the risk that the time needed to execute the task ordering ρ in a random scenario exceeding C_α is less than α. Let $C(q, \rho)$ denote the makespan for scenario q when using task ordering ρ. As mentioned earlier in Section 5.3.1 because we have ρ, $C(q, \rho)$ can be solved quickly in polynomial time as a Linear Program. Also, let $p(C_\alpha, \rho)$ denote the probability that executing the task ordering ρ in a random scenario will need take longer than C_α.

Define the Bernoulli random variable $X_{C_\alpha,\rho}$ as follows:

$$Z_{C_\alpha,\rho} \sim Bernoulli(p(C_\alpha, \rho)). \tag{5.6}$$

We cannot draw samples from $X_{C_\alpha,\rho}$ directly because the probability $p(C_\alpha, \rho)$ is unknown. Instead they are drawn from the nondeterministic function f:

$$f(C_\alpha, \rho, q) = \begin{cases} 1,\ C(q,\rho) > C_\alpha; \\ 0,\ otherwise. \end{cases} \tag{5.7}$$

The nondeterminism comes from the task and transition durations being random variables themselves. Function f is our generative probabilistic model for the random variable $X_{C_\alpha,\rho}$. Drawing samples from this probabilistic model requires drawing sample scenarios from the underlying probabilistic distribution describing the world through the previously mentioned scenario generator and then processing each one by computing $C(q, \rho)$. Separate scenarios can be processed asynchronously and in parallel to increase the speed of running the test using a batch version of the function f:

$$f(C_\alpha, \rho) = Z, \sum_{z \in \{1,\ldots,Z\}} f(C_\alpha, \rho, z), \tag{5.8}$$

where Z can either be set by the user or arbitrarily be chosen by the implementation. Furthermore, when C_α is increased and the SPRT is run again (Algorithm 8 Line 10) then previously computed $C(q, \rho)$ can be memoized and used again without the need for recomputation.

Algorithm 9 shows the pseudocode of the hypothesis test described above. Typically, the SPRT implementation either runs indefinitely before reaching a conclusion or returns a default value when all sampling resources have been used up. For our implementation, the user provides a maximum number of scenarios Q_{max} to be pre-generated and used by the test. As the guarantee on type-II error rates is desired, the algorithm terminates early if there would not be enough evidence to support accepting H_0 even when all remaining scenarios had a makespan $\leq C_\alpha$.

Remark 1. *The log-likelihood ratio thresholds (Equations 5.13-5.14) are approximations primarily designed for situations where θ and β are small [13]. This approximation ensures that the "true" type-I and type-II errors θ' and β' are bounded as $\theta' \leq \frac{\theta}{1-\beta}$ and $\beta' = \frac{\beta}{1-\theta}$.*

Algorithm 9: Testing hypothesis p(C_α, ρ) $\leq \alpha$

Input: reference makespan C_α, a set of task orderings ρ, risk tolerance α, type-I error rate θ, type-II error rate β, maximum number of scenarios Q_{max}

Output:

$\lambda, \zeta, \gamma, l_0, l_1 \leftarrow$ compute using Equations 5.10-5.14

// Compute the maximum number of scenarios with $C^q > C_\alpha$ for H_0 to be accepted with Q_{max}

$a_{Q_{max}} \leftarrow \frac{l_0}{\lambda} + Q_{max}\gamma$ // Total number of scenarios processed

$Z \leftarrow 0$

// Number of scenarios whose makespan is $> C_\alpha$

$Z_\top \leftarrow 0$

while *true* **do**

  // # of scenarios and # of scenarios whose makespan is $> C_\alpha$ since last check

  $z,\ z_\top \leftarrow f(C_\alpha, \rho)$

  if $z = 0$ **then**

    // No more scenarios. Increase C_α and run test again.

    return *false*

  $Z \leftarrow Z + z$

  $Z_\top \leftarrow Z_\top + z_\top$

  $a_0 \leftarrow$ compute using Equation 5.15 for Z scenarios

  if $Z_\top \leq a_0$ **then**

    // Accept H_0 (Accept C_α)

    return *true*

  $a_1 \leftarrow$ compute using Equation 5.16 for Z scenarios

  if $Z_\top \geq a_1$ **then**

    // Accept H_1 (Reject C_α))

    return *false*

  if $Z_\top \geq a_{Q_{max}}$ **then**

    // Unable to accept H_0 even if all remaining scenarios have a makespan $\leq C_\alpha$

    return *false*

It can also be shown that at least one of the inequalities $\theta' \leq \theta$ or $\beta' \leq \beta$ must hold.

5.5 Experimental Evaluations

We empirically evaluated our approach using three sets of experiments in a simulated emergency response domain used in prior work [60, 19, 61, 62, 117]. In this domain, a diverse set of robots with different speeds need to work together to rescue wounded survivors, de-

liver medicine to hospitals, put out fires, and rebuild damaged infrastructure. For all of the experiments, a graph representation of an urban environment named Polypixel (as seen in [141]) was used. CS-HSSRG was set to use $\epsilon = 0.01$, $\theta = 0.05$, and $\beta = 0.05$ for all experiments and we ran all approaches on a desktop with an AMD 3970X CPU, an A6000 GPU, and 16 GB of RAM.

For the first two experiments, we generated a set of 100 HCSTU problems from this domain by randomly sampling the number of robots, survivors, fires, and damaged buildings. Each problem had between 5-15 robots, and 10-30 tasks. We also randomized the locations of the survivors, fires, damaged buildings, hospitals, and the robots' initial location. The specific set of tasks and the allocation of robots to tasks were predetermined for each of these problems.

The first 50 problem instances simulated the possibilities of delays which directly caused temporal uncertainty and we denote this domain as DELAYS. The duration for each task was modeled as $d_i = \hat{d}_i + \mathcal{U}_{[0,300)}$ where $\hat{d}_i$ was the deterministic task duration for τ_i. The durations for transitions were computed from the length of the map edges the robot/coalition must traverse and the speed of the robot/coalition (the speed of a coalition is the speed of the slowest robot in the coalition). When traversing a map edge each robot/coalition had a 5% chance of having a $\mathcal{U}_{[0,60)}$ delay. Note that the task and transition durations were multi-modal since we applied delays to both the task durations and the edge traversal durations.

The second 50 problem instances simulated the possibility of map edges being blocked in the polypixel environment (e.g from debris) which made the traversibility of the environment uncertain. This in turn indirectly created temporal uncertainty as it was unknown what was the best route between two locations on the map which made the transition durations uncertain. For each problem, each map edge had a 10% chance of being blocked. As such the map represented a variant of the Canadian Traveler Problem and we denote this domain as CTP [142]. For these problems, CS-HSSRG used the optimistic rollout policy

from [143] to compute $C(q, \rho)$ for each scenario q in the SPRT. Simliarly to the first 50 problem instances, the distributions for task and transition durations were multi-modal.

For the third experiment, we generated a set of 50 problems STAP-STC problems from the simulated emergency response domain. The problems were randomized similarly to the problems for the first two experiments, however the specific set of tasks and allocation of robots to tasks were not predetermined for these problem instances. The first 25 problem instances used the same model for unexpected delays as described above for the DELAYS domain. The second 25 problem instances used the same model for the possibility of edge blockages as described above for the CTP domain.

5.5.1 Impact of the risk tolerance

The first experiment involved an ablation study to investigate the influence of the risk tolerance α on our approach. For this experiment, we examine our algorithm with different risk tolerance levels ($\alpha \in \{0.1, 0.15, 0.2, 0.25\}$) on the 100 problem instances described previously. In Table 5.1, we present the max proportion of failure $\max(\bar{p}(C_\alpha, \rho))$ and average α-robust makespan C_α. For the proportion of failure $\bar{p}(C_\alpha, \rho)$, we generated 10,000 random scenarios and then computed the makespan for each scenario when executed with the solution task order ρ. The proportion of failure is the proportion of those scenarios whose makespan is larger than the solution C_α and approximates the true risk $p(C_\alpha, \rho)$. The max proportion of failure $\max(\bar{p}(C_\alpha, \rho))$ is the maximum $\bar{p}(C_\alpha, \rho)$ over the 100 problem instances.

Table 5.1: Results for different risk levels

	DELAYS		CTP	
α	$\max(\bar{p}(C_\alpha, \rho))$	C_α	$\max(\bar{p}(C_\alpha, \rho))$	C_α
0.1	0.091	3643.33	0.098	4023.63
0.15	0.136	3528.84	0.122	3881.42
0.2	0.154	3303.83	0.188	3710.10
0.25	0.224	3122.35	0.238	3510.83

In Table 5.1, we observe that as the risk tolerance α increases, the average α-robust makespan tends to decrease. This makes sense as the looser risk tolerance allows CS-HSSRG to ignore more of the slower executing scenarios at the expense of the higher risk of not providing an upperbound. We also observe that the max proportion of failure $\max(\bar{p}(C_\alpha, \rho))$ tends to be close to α which demonstrates that while CS-HSSRG has the theoretical guarantee for its α-robust makespan, it is not just creating overly conserve α-robust makespans to achieve the risk tolerance. Additionally, we observe that CS-HSSRG demonstrates reasonable risk control.

5.5.2 Comparison with other Scheduling Approaches

For our second experiment, we fixed $\alpha = 0.1$ and benchmarked our approach, on the 100 problems described previously, against two state-of-the-art RCPSPU approaches in SORU [121] and the Conditional Value-At-Risk Branch-and-Bound framework from [135] which we will call CVAR-BNB, as well as the solving the Sample Average Approximation (SAA) formulation from Equation 5.3 directly. To generalize the problem to the RCPSPU for SORU and CVAR-BNB, we represented the individual robots as resources and each task or transition duration is represented as an activity. Each approach was given a 60s timeout for each problem instance. If an approach hit the timeout then the best feasible approach it generated was used.

We compared all four approaches on several metrics including the percentage of solutions where a temporal constraint was violated, the percentage of solutions where $\bar{p}(C_\alpha, \rho) > \alpha$, the percentage of successful solutions ($\bar{p}(C_\alpha, \rho) \leq \alpha$), the average computation time, the percentage of the problems where the approach hit the 60s timeout, the average $\bar{p}(C_\alpha, \rho)$, and the average C_α. The average C_α was only computed over problems in which all four approaches computed solutions with $\bar{p}(C_\alpha, \rho) \leq \alpha$.

Table 5.2: Summary of comparison results ($\alpha = 0.1$). Red indicates the best result for the row.

	DELAYS				CTP			
	Ours	SAA	SORU [121]	CVAR-BNB [135]	Ours	SAA	SORU	CVAR-BNB
TC Failure (%)	0	0	8[a]	0	0	0	12[1]	0
$\bar{p}(C_\alpha, \rho) > \alpha$ (%)	0	40	0	28	0	68	0	50
Success (%)	100	60	92	72	100	32	88	50
Avg. $\bar{p}(C_\alpha, \rho)$	0.079 ± 0.020	0.097 ± 0.053	0.003 ± 0.009	0.088 ± 0.043	0.071 ± 0.016	0.112 ± 0.029	0.011 ± 0.010	0.096 ± 0.045
Avg. C_α [b]	3794.71 ± 1305.60	3376.19 ± 1953.00	4554.90 ± 1436.70	3679.04 ± 2390.73	4365.24 ± 1620.32	4268.57 ± 1800.43	5813.46 ± 2132.32	4316.96 ± 2532.38
Avg Comp. Time (s)	0.822 ± 0.426	34.057 ± 24.288	49.760 ± 17.154	8.313 ± 1.913	1.014 ± 0.34	42.02 ± 19.88	57.64 ± 9.41	10.93 ± 8.31
Timeout (%) [c]	0	10	64	2	0	16	82	2

[a] SORU uses hard resource constraints and soft temporal constraints, which sometimes allow it to violate the temporal constraints.

[b] Computed using trials for which all four approaches' $\bar{p}(C_\alpha, \rho) \leq \alpha$

[c] Upon hitting the 60s timeout, a feasible solution was retrieved if one had been found by the approach.

SAA

As can be seen in Table 5.2, SAA did not violate any temporal constraints, but generated solutions that violated the risk tolerance 40% of the time for the DELAYS problems and 68% of the time for the CTP problems. It also had an average proportion of failure $\bar{p}(C_\alpha, \rho)$ of 0.097 and 0.112 respectively, which were either close or over the desired risk tolerance. These risk tolerance violations were likely because SAA considers the α-quantile samples but does not get to choose representative samples like CS-HSSRG and additionally does not use statistical testing to guarantee its "α-robust makespan" is actually α-robust. On the other hand, CS-HSSRG used its heuristic sample selection and the SPRT to successfully generate risk tolerant solutions on 100% of the problems for both domains. SAA did on average compute a smaller C_α than any of the other approaches including CS-HSSRG. On average for these problems, its C_α was 90% of the C_α generated by CS-HSSRG. As SAA frequently violated the risk tolerance, it is likely that when it doesn't violate the risk tolerance the C_α generated has a less conservative risk resulting in the lower average C_α. When computing a solution, SAA on average took 41.4x the time to compute solution as CS-HSSRG did. Additionally, it hit the timeout on 13% of problems in total.

SORU

SORU was the only approach that generated solutions which violated temporal constraints. This is caused by SORU using soft temporal constraints. It did not have any risk tolerance violations, which is caused by its extremely conservative solutions with an average risk of 0.003 and 0.011 respectively for the two domains. This extremely conservative risk resulted in it having the highest average C_α. The average C_α generated by SORU was 120% of the C_α generated by CS-HSSRG. SORU also took the longest of the four approaches on average and took on average 60.5x as long to compute a solution as CS-HSSRG. Additionally, it hit the 60s timeout on 73% of all the problems.

CVAR-BNB

Similar to SAA, CVAR-BNB did not violate any temporal constraints, but violated the risk tolerance on some of the problems (28% and 50% respectively). The risk tolerance violations are likely because CVAR-BNB does not get to choose representative samples like CS-HSSRG and additionally does not use statistical testing to guarantee its "α-robust makespan" is actually α-robust. CVAR-BNB on average generated C_α that was 97% of the C_α generated by CS-HSSRG, however on average it took 10.1x as long to compute its solution. Also, while it was quicker than both SAA and SORU on average, it did hit the 60s timeout on 2% of the total problems.

Ours

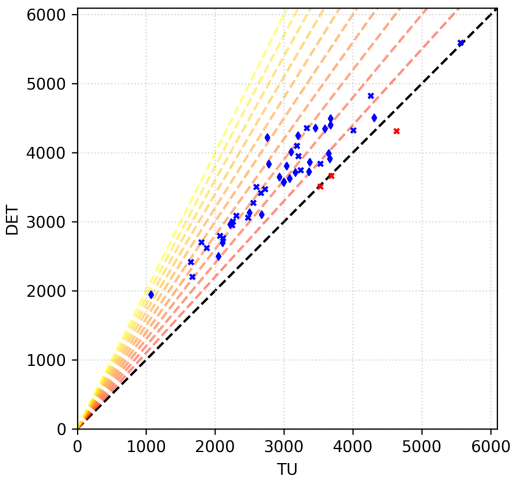
In comparison to the other three approaches, CS-HSSRG was the only approach to never have a temporal constraint violation or a risk tolerance violation and was the only approach to never hit the 60s timeout. Additionally, it on average generated C_α that were within 10% of SAA, 3% of CVAR-BNB, and were 20% faster than SORU.

Furthermore, CS-HSSRG was an order of magnitude faster than all other approaches (on average 41.4x faster than SAA, 60.5x faster than SORU, and 10.1x faster than CVAR-

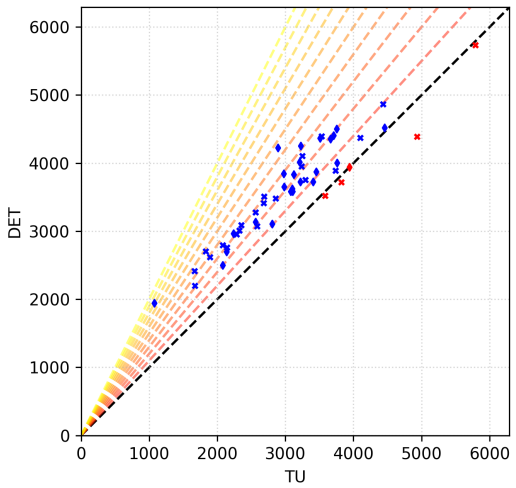
BNB). This demonstrates an efficient approach that produces solutions with similar quality to state-of-the-art and a risk tolerance guarantee.

5.5.3 Usage within the GRSTAPS Framework

For our third experiment, we demonstrated how CS-HSSRG's ability to reason about uncertainty and risk aids the ability of our heterogeneous multi-robot coordination framework GRSTAPS to make decisions about what tasks to select and who to allocate to those tasks. We used CS-HSSRG as the scheduling layer of GRSTAPS by using the α-robust makespan to replace the normal makespan in the Normalized Schedule Quality (NSQ) heuristic and as the path cost for the task planning layer. We compared this variant of GRSTAPS which we will called GRSTAPS^{TU} against the original GRSTAPS which we called GRSTAPS^{DET}. For each problem, GRSTAPS^{DET} was provided a determinized version of the problem where $d_i = \mathbb{E}[d_i]$, $\phi_{ij} = \mathbb{E}[\phi_{ij}]$, and $\phi_i = \mathbb{E}[\phi_i]$.



(a) Average makespan when using the solution task ordering ρ.



(b) Sum of C^ρ and computation time, where C^ρ is the average makespan when using the solution task ordering ρ.

Figure 5.1: The blue (red) markers are problem instances where GRSTAPS^{TU} did better (worse) than GRSTAPS^{DET}. Each successive dashed colored line is the result from GRSTAPS^{DET} being 10% worse than the result from GRSTAPS^{TU} (i.e. the first line represents 10% worse, the second 20% worse, etc).

We measured the quality of the solution task orderings ρ produced by each approach.

For each problem instance, we generated 10,000 random scenarios and then computed the makespan for each scenario when executed with the solution task ordering. We then took the average makespan over the 10,000 scenarios. The average makespans can be seen in Figure 5.1a. In Figure 5.1b, we show the sum of the average makespans and the time needed to compute the solution. For both figures, the blue markers are problem instances where GRSTAPS^{TU} did better than GRSTAPS^{DET}. The red markers are the opposite. The black dashed line shows where both approaches had equal results. Each successive dashed colored line is the result from GRSTAPS^{DET} being 10% worse than the result from GRSTAPS^{TU} (i.e. the first line represents 10% worse, the second 20% worse, etc). Additionally, we use the makespans computed for these 10,000 realizations to compute $\bar{p}(C, \rho)$ which is shown in Table 5.3.

Table 5.3: Results for $\bar{p}(C, \rho)$

	DELAYS	CTP
DET	0.5275 ± 0.3271	0.7805 ± 0.2175
TU	0.0858 ± 0.0061	0.0864 ± 0.0063

As can be seen, by reasoning about uncertainty, GRSTAPS^{TU} was able to create better task orderings that on average result in better makespans. GRSTAPS^{DET} uses the expected values of individual distributions while attempting to create a solution that minimizes makespan, however, it does not consider anything more about the distribution such as variance or modality. This can cause it to select tasks, allocations, and task orderings that have low expected values for duration, but have a high risk of having large durations. Due to the large number of decisions it has to make (which tasks, which robots, which task orderings), there are a lot of possibilities for the individual risks of a task or transition taking longer than expected to be realized in a scenario and can result in high overall risk. This can be seen in Table 5.3 where GRSTAPS^{DET} has high average and standard deviation for the estimate of overall risk $\bar{p}(C, \rho)$. On the other hand while CS-HSSRG does not directly consider the distributions, it indirectly considers the distributions through the sampling and

sample selection method and then it directly considers the risk through the SPRT. As both approaches generated and expanded similar numbers of nodes for both task planning and task allocation layers (less than 0.1% difference on average), it is likely that the improvement in solution quality is a result of CS-HSSRG providing better search guidance to both the task planning and task allocation layers of GRSTAPS. This better guidance results in higher quality, low risk solutions.

5.5.4 Limitations

Although CS-HSSRG provides significant performance gains for HCSTU problems relative to existing methods, there are some limitations. The user must determine an acceptable risk tolerance α, the number of scenarios to be used in the S-SAA MILP η, and the number of scenarios in F. As mentioned earlier, increasing η and the size of F increases the quality of the solution and the likelihood that the risk tolerance is respected when running the first instance of the SPRT, however, they also increase the computation time needed to solve the problem. As such, when selecting these parameters there is a trade-off between solution quality and efficiency. For the experiments, we empirically determined these η and the size of F through some preliminary experiments and measuring the number of SPRT failures; however, future work may determine a faster method to automatically select them.

Furthermore, there is extra reasoning required to consider uncertainty and risk which comes at the cost of lower efficiency and higher computation times as GRSTAPS^{TU} on average took an order of magnitude longer to compute a solution than GRSTAPS^{DET} for the same problem instance. As both approaches generated and expanded similar numbers of nodes for both task planning and task allocation, this is almost entirely caused by CS-HSSRG being slower to compute than solving the Deterministic MILP from Equation 5.1 as GRSTAPS^{DET} does. This is expected as Equation 5.5 has roughly $\eta\times$ as many variables and constraints as Equation 5.1 and the addition of the sample selector heuristic and the SPRT. However, even with this increase in computation time, on average the total time

needed to both compute a solution and execute the resulting solution is less when using GRSTAPS^{TU} than GRSTAPS^{DET} for 90% of the problem instances.

5.6 Discussion & Conclusion

We introduced a novel sampling-based risk-aware approach named Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee (CS-HSSRG) to solve the Heterogeneous Coalition Scheduling with Temporal Uncertainty (HCSTU) Problem. This approaches includes a heuristic method for selecting representative scenarios and utilizes the Sequential Probability Ratio Test to guarantee the solution it generates is actually α-robust. We first demonstrate that CS-HSSRG has reasonable risk control and does not just generate overly conservative solutions while respecting the risk tolerance. Then we demonstrate that CS-HSSRG is more efficient than a Sample Average Approximation baseline and two state-of-the-art approaches while generating comparable or better quality solutions. Furthermore, it was the only one of the approaches to not have a single temporal constraint violation or risk tolerance violation. Finally, we demonstrate how reasoning about uncertainty and risk tolerance aid in guiding a higher-level multi-robot coordination framework in GRSTAPS to higher quality, low risk solution.

However, CS-HSSRG does have its limitations and there are open questions left to explore. First, it assumes access to either a model of uncertainty or a generator that can generate samples of possible task and task transition durations. While this is generally accepted as a reasonable assumption in literature [121, 135], there may be environments for which this assumption does not hold. Those type of environments may need some combination of pre-operative and intra-operative methods. Second, a user must decide on the number of samples to be utilized by CS-HSSRG. As mentioned previously, using more samples increases the quality of the solution at the expense of decreasing the computational efficiency of the algorithm. For our experiments, we decided on the number of samples through preliminary experiments, but in future work it is possible to develop

a method to automatically select the number of samples needed perhaps through analysis on the temporal graph or through learning. Third, we currently utilize a simple domain-independent heuristic to label the samples to select what samples CS-HSSRG will utilize in the S-SAA MILP. It is possible that a more sophisticated heuristic that either performs more analysis on the temporal graph or is learned could result in higher quality solutions with lower α-robust makespans while still satisfying the desired risk tolerance. Finally, CS-HSSRG is a pre-operative method for handling temporal uncertainty. As such, it does not benefit from information that is discovered during the execution of the solution. Pairing CS-HSSRG with an intra-operative method that can utilize information discovered during execution to improve the schedule will likely result in higher quality execution that takes less time.

5.7 Appendix: Sequential Probability Ratio Test

The Sequential Probability Ratio Test (SPRT) [13] is a sequential hypothesis test. As such, computations needed to confirm or reject the hypothesis can be lazily applied to samples as needed instead of drawing them all in advance. In this section, we give a brief overview of the application of the SPRT to the binomial distribution, which is the one we leverage in our work as shown in Section 5.4.3, and refer the reader to the original paper by Wald [13] for a more complete treatment.

In the binomial case, a sample can be classified into two categories: 0 and 1. Let p be the probability that a sample belongs to category 0. We deal with the problem of testing the hypothesis that p does not exceed a given value p' against the alternative hypothesis that $p > p'$. A SPRT for the binomial case can be defined by specifying a null hypothesis $H_0 \stackrel{\text{def}}{=} p = p_0$, with $p_0 < p'$, and an alternative hypothesis $H_1 \stackrel{\text{def}}{=} p = p_1$, with $p_1 > p'$. The small interval (p_0, p_1) is called the *indifference region* — the SPRT requires that we are okay with accepting either hypothesis when the true value of p is in the indifference region. Define the function $f(q)$:

$$f(q) = \begin{cases} 1, & \textit{the sample } q \textit{ belongs to category 1}; \\ 0, & otherwise. \end{cases} \tag{5.9}$$

Given the desired type-I and type-II error rates θ[2] and β, prior to the test the following values can be calculated:

$$\lambda = \log \frac{p_1}{p_0} - \log \frac{1-p_1}{1-p_0} \tag{5.10}$$

$$\zeta = \log \frac{1-p_0}{1-p_1} \tag{5.11}$$

$$\gamma = \frac{\zeta}{\lambda} \tag{5.12}$$

$$l_0 = \log \frac{\beta}{1-\theta} \tag{5.13}$$

$$l_1 = \log \frac{1-\beta}{\theta} \tag{5.14}$$

The test is given as follows. At the z^{th} observation, calculate the two quantities

$$a_0 = \frac{l_0}{\lambda} + z\gamma \tag{5.15}$$

and

$$a_1 = \frac{l_1}{\lambda} + z\gamma. \tag{5.16}$$

Let $z_\top$ denote the number of observations where $f(\cdot) = 1$ in the first z samples inspected. For each time a_0 and a_1 are calculated, there are three possible outcomes:

1. accept H_0 if $z_\top \leq a_0$,
2. accept H_1 if $z_\top \geq a_1$,
3. or more observations are needed if $a_0 < z_\top < a_1$.

[2]In Wald's paper [13] he uses α as the type-I error rate. We use θ to represent the type-I error rate here due to α being used for the α-robust makespan.

The third outcome means that the samples drawn so far do not provide enough evidence either for accept or rejecting the null hypothesis H_0. More samples need to be drawn, a_0 and a_1 need to be recomputed, and the conditions need to be checked again. The SPRT does not guarantee an upper-bound on the number of samples needed to reach a conclusion; in order to retain the guarantees on error rates, the above process must be repeated until a conclusion can be reached.

CHAPTER 6
LEARNING TO SET TASK ORDERINGS FOR HETEROGENEOUS COALITION SCHEDULING WITH DEADLINES

6.1 Introduction

In this chapter, we transition away from exploring temporal uncertainty and explore a different type of temporal constraint within the context of the scheduling sub-problem of the Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) problem we formalized in Chapter 4[1]. We focus this chapter on exploring deadlines as an additional temporal constraint. Furthermore, we explore the use of a learning-based approach to solve this new problem.

Significant effort has been invested into finding analytical solutions to multi-robot scheduling problems that balance optimality, completeness, and computational efficiency [144, 145, 146]. However, these problems are generally NP-hard [137] and have several combinatoric factors (i.e., number of tasks, number of robots, etc). As such, solving the problem exactly is computationally expensive and does not scale to large problems [147]. On the other hand, hand-crafted heuristic approaches are difficult to design and require domain-specific knowledge that does not generalize to other domains [148, 147].

Data-driven approaches have been presented for a number of multi-robot coordination sub-problems (e.g. Multi-Vehicle Routing, Task Allocation, Communication Networks, etc) and have demonstrated the ability to find near-optimal solutions to NP-Hard problems [146]. In particular, Graph Neural Networks (GNNs) have demonstrated notable performance and generalize well from small-scale problems to large-scale problems for these

[1]The material in this chapter is based on:
A. Messing and S. Hutchinson, "Learning to Set Task Orderings for Heterogeneous Coalition Scheduling with Deadlines," IEEE Transactions on Robotics, 2023.

multi-robot coordination sub-problems [149, 146, 150, 147, 151]. Recently, GNNs have been applied to heterogeneous graphs increasing the expressiveness of the model and the complexity of the problem that can be solved [147].

In this chapter, we introduce a Heterogeneous Gated Graph Convolution Network (HG-GCN) model that learns the probability that pairs of tasks are ordered in a specific way in an optimal schedule. This model takes as input a heterogeneous graph that represents the relationships between time points and can encode all 13 of Allen's temporal relationships [14].

Furthermore, we introduce a search-based approach, named Heterogeneous Temporal Graph Scheduler (HTGS), with two search strategies that utilize the output of the heterogeneous gated graph convolution network model to approximately solve the Heterogeneous Coalition Scheduling with Deadlines Problem. HTGS uses the probabilities from the HG-GCN model to incrementally construct a Simple Temporal Network (STN) [33] by setting task orderings until a solution schedule that respects the temporal constraints is generated. Figure 6.1 illustrates the proposed scheduling algorithm.

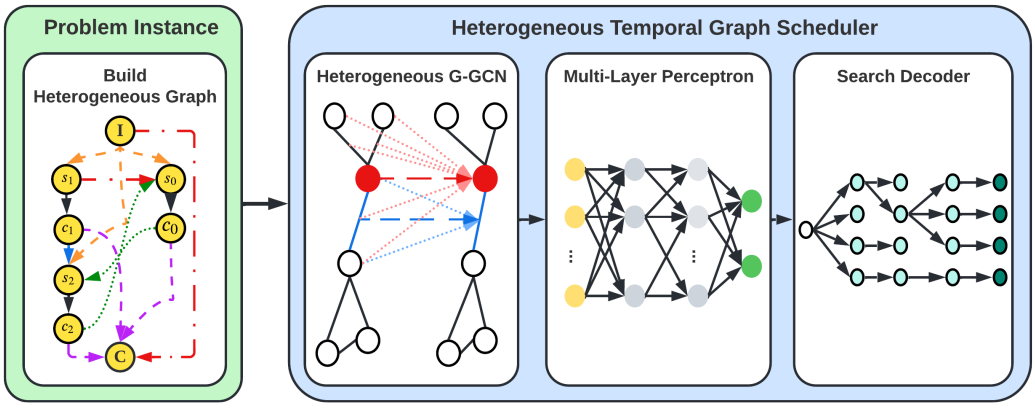


Figure 6.1: The proposed Heterogeneous Temporal Graph Scheduler approach for solving the Heterogeneous Coalition Scheduling with Deadlines Problem. The Heterogeneous G-GCN block is based on an image from [152]

.

To illustrate the impact of our contributions, we evaluate the performance of this approach in a simulated emergency response domain. First, we use an ablation study to select the best-performing model to use in our approach. Second, we compare the HTGS to state-

of-the-art heterogeneous multi-robot scheduling algorithms and an optimal Mixed-Integer Linear Programming (MILP) baseline. Third, we integrate the approach into our heterogeneous multi-robot coordination framework Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling (GRSTAPS) [117] and evaluate the improvement in the performance of the overall framework on the Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) Problem. The results of these evaluations demonstrate the efficacy of our new approach.

6.2 Related Work

This paper lies at the intersection of a traditional robotics research avenue in Multi-Robot Scheduling and a more recent machine learning one in Graph Neural Networks. They are briefly discussed below relative to the concepts that appear in this paper.

6.2.1 Multi-Robot Scheduling

A rich body of work has addressed the Multi-Robot Scheduling Problem [44] and the closely related Multi-Robot Task Allocation (MRTA) Problem [42, 43]. Recent survey papers [42, 43, 44] have summarized MRTA techniques and solutions based on optimization approaches (e.g., exact, heuristic, metaheuristics), AI approaches (e.g., constraint programming, multi-robot consensus, market-based, game-theoretic models, machine learning), and combinations of these approaches. Based on the common taxonomies from these survey papers, our work falls under the XD[ST-MR-TA-SP] categorization where an individual robot can only participate in a single task at a time (ST), but multiple robots can form coalitions and collectively executable a single task (MR). The XD[ST-MR-TA-SP] problem considers cross-schedule dependencies (XD), where the effective utility of a robot for a task depends not only on its own schedule but also on the schedules of other robots in the system, and represents temporal relationships between tasks through Synchronization and Precedence Constraints (SP). Furthermore, we consider a variant of this problem with

multiple species of robots and temporal deadlines.

Koes *et al.* [56] present Constraint Optimization Coordination Architecture, an anytime algorithm that utilizes a commercial Mixed-Integer Linear Programming (MILP) Solver on a novel declarative formulation, however, this approach was only demonstrated with 5 robots and 15 goals. Zhang *et al.* [122] introduce four heuristics to solve the heterogeneous coalition scheduling problem, however, they do not consider precedence constraints. Bischoff *et al.* [123] propose a two-step approach that utilizes a construction heuristic and an improvement heuristic based upon the relocate neighborhood operator commonly used in vehicle routing; however, they only demonstrate results with 3 robots and do not include deadlines. Neville *et al.* [5], as a subset of their Trait-Based Time-Extended Task Allocation Problem, study a similar scheduling problem to ours and discuss a Simple Temporal Network (STN) [33]-based scheduling approach that utilizes a Tabu Search [59] through a space of task orderings; however, their problem does not include deadlines. In their later work in [153], they introduce a MILP-based scheduling algorithm. While each of these approaches has their own respective advantages, prior research does not address the need to efficiently solve large-scale problems with heterogeneous robots, coalitions, tightly-coupled tasks, and deadlines.

6.2.2 Graph Neural Networks

Graph Neural Networks (GNNs) are a class of neural network architectures that are specifically designed to learn from the structure and properties of graph data [154]. For a complete overview of this rapidly expanding subfield of machine learning, we refer the interested reader to Wu *et al.* [155] for a survey, and to Dwivedi *et al.* [156] for a recent GNN benchmarking paper.

Recently, a number of works have used GNNs to learn policies for various components of solving task allocation and scheduling problems (e.g., Traveling Salesman Problem, Vehicle Routing Problem, Communication and Teaming, etc). Banfi *et al.* [141] present a

hierarchical planner that utilizes a GNN to produce a heuristic to estimate subteam performance for specific coalitions on specific routing tasks. Blumenkamp *et al.* [146] develop a system for decentralized execution of GNN-based communication policies. Wang *et al.* [147] propose a novel Heterogeneous Graph Attention Network model that learns a policy to append an unscheduled task to the end of the partial schedule of a robot. This approach shares some similarities with ours as they also encode a heterogeneous multi-robot scheduling problem as a heterogeneous graph; however, our problem includes multi-robot tasks, includes task duration and transition times based on the time needed to execute motion plans, and assumes that an allocation has already been assigned. Additionally, the two approaches utilize the GNNs in different ways - they are incrementally constructing individual fully-ordered schedules for each robot and we use a GNN to generate probabilities that each task ordering is set in a partially-ordered plan/schedule.

6.3 Problem Description

Consider a heterogeneous team of K robots that must collectively execute N tasks. Each task must be executed by one or more robots (i.e. a coalition), but each robot can only participate in a single task at a time (ST-MR-TA).

In our problem, a task τ_i is defined by a duration d_i as the amount of time needed to executeτ_i by the robots that are assigned to it, an initial transition ϕ_i as the minimum amount of time needed for all of the robots assigned to τ_i to reach the initial configuration of τ_i from their individual initial configurations, the time point for which the task starts (s_i), and the time point for which the task is completed ($c_i = s_i + d_i$). Tasks are temporally constrained by a set of precedence constraints $\mathcal{P}$, a set of mutex constraints $\mathcal{M}$, a set of relative deadlines $\mathcal{T}_{rel}$, and a set of absolute deadlines $\mathcal{T}_{abs}$.

A *precedence constraint ($\tau_i \prec \tau_j$)* is a temporal relationship between two tasks τ_i and τ_j that requires that the execution of τ_i concludes before τ_j starts. Each precedence constraint $\tau_i \prec \tau_j$ has a transition duration ϕ_{ij} that defines the minimum amount of time needed for

all robots assigned to both τ_i and τ_j to travel from the terminal configuration of τ_i to the initial configuration of τ_j. If there are no robots that are assigned to both τ_i and τ_j then $\phi_{ij} = 0$.

If a robot is assigned to multiple tasks, a mutex constraint is created between each pair of tasks that the robot is assigned to participate in. A *mutex constraint ($\tau_i \leftrightarrow \tau_j$)* between two tasks, τ_i and τ_j, represents the disjunction that either τ_i must conclude before τ_j starts ($\tau_i \prec \tau_j$) or τ_j must conclude before τ_i starts ($\tau_j \prec \tau_i$). If a robot is assigned to two tasks τ_i and τ_j where there already exists a precedence constraint (e.g. $\tau_i \prec \tau_j$) then the precedence constraint supersedes the mutex constraint as there is no decision to be made on the ordering of the two tasks. As such, $\mathcal{P} \cap \mathcal{M} = \emptyset$. Each mutex constraint $\tau_i \leftrightarrow \tau_j$ has a pair of transition durations ϕ_{ij} and ϕ_{ji} that define the minimum amount of time needed for all robots assigned to both τ_i and τ_j to travel from the terminal configuration of τ_i to the initial configuration of τ_j and to travel from the terminal configuration of τ_j to the initial configuration of τ_i respectively.

There are two types of deadlines represented in this problem: absolute deadlines and relative deadlines. An *absolute deadline* constrains the upper bound between a time point and when time is zero. An absolute deadline is represented by the tuple $(p_i,\ t)$ where p_i is equal to either s_i if the deadline is on the start of task τ_i or c_i if the deadline is one completion of task τ_i and t is the upper bound on the time point that p_i is equal to. A *relative deadline* is a temporal constraint on the upper bound of the temporal difference between two time points. A relative deadline is represented by the tuple $(p_i\ p_j,\ t)$. Similarly to the absolute deadline, p_i (p_j) is equal to either s_i (s_j) if the deadline is on the start of task τ_i (τ_j) or c_i (c_j) if the deadline is on the completion of task τ_i (τ_j). t is the upper bound of the temporal difference between the two time points p_i and p_j ($t \geq p_j - p_i$).

The Heterogeneous Coalition Scheduling with Deadlines (HCSD) problem is defined by

- a set of task durations $\mathcal{D} = \{d_i \mid i \in \mathcal{I}\}$,

- a set of initial task transitions $\Phi_{init} = \{\phi_i \mid i \in \mathcal{I}\}$
- a set of precedence constraints $\mathcal{P} \subseteq \mathcal{I}^2$,
- a set of mutex constraints $\mathcal{M} \subseteq \mathcal{I}^2$,
- a set of relative deadlines $\mathcal{T}_{rel} \subseteq \{s_{pred}, c_{pred}\} \times \{s_{suc}, c_{suc}\} \times \mathcal{I}^2 \times \mathbb{R}^+$,
- a set of absolute deadlines $\mathcal{T}_{abs} \subseteq \{s, c\} \times \mathcal{I} \times \mathbb{R}^+$,
- and a set of task transition durations $\Phi = \{\phi_{ij} \mid (i,j) \in \mathcal{P}\} \cup \{\phi_{ij} \mid (i,j) \in \mathcal{M}\} \cup \{\phi_{ji} \mid (i,j) \in \mathcal{M}\}$

where $\mathcal{I}$ is the index set from 1 to N, s_{pred}/c_{pred} signifies whether the deadline is relative to the start or completion time point of the predecessor task, and s_{suc}/c_{suc} signifies whether the deadline is relative to the start or completion time point of the successor task.

We provide a formal definition of this problem as the following Mixed-Integer Linear Programming (MILP) model:

$$min \;\; C \tag{6.1a}$$

$$s.t. \;\; C \geq c_i \qquad \forall i \in \mathcal{I} \tag{6.1b}$$

$$s_j \geq c_i + \phi_{ij} \qquad \forall (i,j) \in \mathcal{P} \tag{6.1c}$$

$$s_j \geq c_i + \phi_{ij} - M(1 - \delta_{ij}) \qquad \forall (i,j) \in \mathcal{M} \tag{6.1d}$$

$$s_i \geq c_j + \phi_{ji} - M\delta_{ij} \qquad \forall (i,j) \in \mathcal{M} \tag{6.1e}$$

$$s_i \geq \phi_i \qquad \forall i \in \mathcal{I} \tag{6.1f}$$

$$t \geq p_i \qquad \forall (p_i, t) \in \mathcal{T}_{abs} \tag{6.1g}$$

$$t \geq p_j - p_i \qquad \forall (p_i, p_j, t) \in \mathcal{T}_{rel} \tag{6.1h}$$

$$\delta_{ij} \in \{0, 1\} \qquad \forall (i,j) \in \mathcal{M} \tag{6.1i}$$

where δ_{ij} is a boolean indicator that is 1 when the mutex constraint between the i^{th} and j^{th} tasks ($\tau_i \leftrightarrow \tau_j$) has been reduced to the precedence constraint from the i^{th} task to the

j^{th} task ($\tau_i \prec \tau_j$) and 0 when the mutex constraint has been reduced to the precedence constraint from the j^{th} task to the i^{th} task ($\tau_j \prec \tau_i$) and M is a large constant used to enforce conditional constraints.

Constraint 6.1a is the optimization objective of minimizing the makespan. Constraint 6.1b ensures that the makespan represents the time that the last task is finIshed. Constraint 6.1c applies the precedence constraints and includes the time the robots assigned to tasks τ_i and τ_j need to transition from τ_i to τ_j. Constraints 6.1d and 6.1e apply the disjunction of the mutex constraints. Depending on the value of the mutex indicator variable (δ_{ij}) only one of the two equations is used by the optimization for each mutex constraint. This disjunction is what makes this an NP-Hard problem [137]. Once each of the mutex indicator variables is set then the problem becomes a Linear Problem and can be solved in polynomial time [33]. Constraint 6.1f constrains each task so that all robots assigned to the task can reach the initial configuration of the task from their individual initial configurations before the task starts. Constraint 6.1g applies the absolute deadlines ensuring that a task cannot complete (start) before a specified time. Constraint 6.1h applies the relative deadlines ensuring that the temporal difference between two specified time points is not more than a specified time. Finally, Constraint 6.1i constrains the mutex indicator variables to be boolean variables.

The solution to the HCSD problem is the set of task orderings $\rho = \{\delta_{ij} \mid (i,j) \in \mathcal{M}\}$ that minimizes the makespan C (i.e., the total time needed to execute all of the tasks while considering the task durations, transition durations, and various temporal constraints). The Heterogeneous Coalition Scheduling with Deadlines problem is a subset of the Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) Problem from our prior work [117].

6.4 Building a Heterogeneous Graph

To utilize a Graph Neural Network model as part of our approach, we need to encode the constraints from the problem as a heterogeneous graph. A heterogeneous graph $\mathcal{G} =$

$\{\mathcal{V} = \{\mathcal{V}_1, \ldots, \mathcal{V}_n\},\ \mathcal{E} = \{\mathcal{E}_1, \ldots, \mathcal{E}_m\}\}$ is a graph with n different types of vertices and m different types of edges where $n + m > 2$. In our heterogeneous graph, there is a singular vertex type where all vertices represent a time point and there are four edge types representing four different types of temporal differences: lower bound, upper bound, exact, and mutex. The mutex edges also represent a lower bound on the temporal difference between two time points; however, for each mutex constraint there are two edges and the goal of the network is to determine which of the two edges is better for minimizing the makespan of the overall schedule. As such, they are considered a different edge type.

The algorithm for constructing the heterogeneous graph used as input to our model is shown in Algorithm 10. In line 1, we initialize the vertex set and each of the four edge sets. Then in lines 2 and 3, we create a vertex for the initial state (v_I) and a vertex for the makespan (v_C). In lines 5 and 6, we create two vertices for each task τ_i with one representing the start time point for the task (v_{s_i}) and one representing the completion time point for the task (v_{c_i}). In line 7, we connect start and completion time points for each task with an exact edge that is weighted by the duration of the task. In line 8, we then connect the initial state vertex (v_I) with each of the task start vertices (v_{s_i}) with a lower bound edge weighted by the initial transition duration ϕ_i for the task τ_i. In line 9, we connect each of the task completion vertices with the makespan vertex with a lower bound edge weighted with 0. At this point, for each task τ_i there is a path from the initial state vertex v_I to the start time point vertex v_{s_i} to the completion time point vertex v_{c_i} to the makespan vertex v_C. Next, we add edges representing the relationships between tasks. In lines 10 and 11, we add lower bound edges representing the precedence constraints which are weighted by their respective transition durations. In lines 12-14, for each mutex constraint ($\tau_i \leftrightarrow \tau_j$) we add two mutex edges each weighted by its respective transition duration. One connects v_{c_i} to v_{s_j} and the other connects v_{c_j} to v_{s_i} Lines 15 and 16 add upper bound edges representing the absolute deadlines weighted by the value of the deadline. These edges connect v_I to each bounded time point vertex. Finally, lines 17 and 18 add upper bound edges representing the

Algorithm 10: Building a heterogeneous graph input

Input: $\mathcal{I}$, $\mathcal{D}$, $\mathcal{P}$, $\mathcal{M}$, $\mathcal{T}_{abs}$, $\mathcal{T}_{rel}$
Output: $\mathcal{G}$
// Initialize vertex and edge sets
$\mathcal{V} = \{\}; \mathcal{E}_{lb} = \{\}; \mathcal{E}_{ex} = \{\}; \mathcal{E}_{ub} = \{\}; \mathcal{E}_{mux} = \{\}$
// Create vertex representing the initial state
$\mathcal{V} \leftarrow v_I$
// Create vertex representing the makespan
$\mathcal{V} \leftarrow v_C$
for $i \in \mathcal{I}$ **do**
    // Create vertex for the start of τ_i
    $\mathcal{V} \leftarrow v_{s_i}$
    // Create vertex for the finish of τ_i
    $\mathcal{V} \leftarrow v_{c_i}$
    // Add edge representing the duration
    $\mathcal{E}_{ex} \leftarrow \{v_{s_i}, v_{c_i}, d_i\}$
    // Add edge representing the initial transition
    $\mathcal{E}_{lb} \leftarrow \{v_I, v_{s_i}, \phi_i\}$
    // Add edge connecting task completion to the makespan
    $\mathcal{E}_{lb} \leftarrow \{v_{c_i}, v_C, 0\}$
// Add edges representing the precedence constraints
for $\{i, j, \phi_{ij}\} \in \mathcal{P}$ **do**
    $\mathcal{E}_{lb} \leftarrow \{v_{c_i}, v_{s_j}, \phi_{ij}\}$
// Add edges representing the mutex constraints
for $\{i, j, \phi_{ij}, \phi_{ji}\} \in \mathcal{M}$ **do**
    $\mathcal{E}_{mux} \leftarrow \{v_{c_i}, v_{s_j}, \phi_{ij}\}$
    $\mathcal{E}_{mux} \leftarrow \{v_{c_j}, v_{s_i}, \phi_{ji}\}$
// Add edges representing the absolute deadlines
for $\{p_i, t\} \in \mathcal{T}_{abs}$ **do**
    $\mathcal{E}_{ub} \leftarrow \{v_I, v_{p_i}, t\}$
// Add edges representing the relative deadlines
for $\{p_i, p_j, t\} \in \mathcal{T}_{rel}$ **do**
    $\mathcal{E}_{ub} \leftarrow \{v_{p_i}, v_{p_j}, t\}$
$\mathcal{G} = \{\{\mathcal{V}\}, \mathcal{E} = \{\mathcal{E}_{lb}, \mathcal{E}_{ex}, \mathcal{E}_{ub}, \mathcal{E}_{mux}\}\}$
return $\mathcal{G}$

relative deadlines weighted by the value of the deadline. An example of this heterogeneous graph is shown in Figure 6.2.

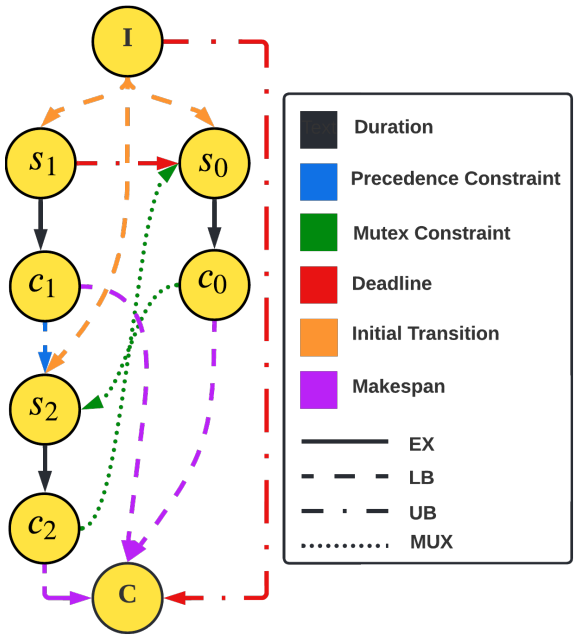


Figure 6.2: Example input heterogeneous graph

6.5 Model

In this section, we describe the components of the GNN model utilized by our approach (See Figure 6.1). This model, given the heterogeneous graph that we built in the previous section, outputs the probability for each task ordering δ_{ij} to be either 1 or 0 in the optimal solution. The model consists of an embedding layer, a series of Heterogeneous Gated Graph Convolution Layers (HG-GCLs) followed by a Multi-Layer Perceptron (MLP) Classifier.

Input Embedding Layer

To embed the input node features to h-dimensional features, we use a linear layer:

$$x_i' = W_0^x x_i + b_0^x$$

Similarly, we embed each of the different input edge-type features to h-dimensional features through edge-type specific linear layers where ψ represents the edge-type:

$$e_{ij}^{\psi\prime} = W_0^{\psi} e_{ij}^{\psi} + b_0^{\psi}$$

Heterogeneous Gated Graph Convolution Layer

For this layer, we leverage the Gated Graph Convolution Network (G-GCN) architecture introduced by Bresson and Laurent in [152] with the additional edge feature representation and the dense attention map added in [157]. This network was chosen as the basic building block for our architecture because it was recently shown by Dwivedi *et al.* [156] to outperform several other architectures at both graph regression and Traveling Salesman Problem solution generation. Furthermore, unlike other anisotropic GNN architectures, it explicitly maintains edge features at each layer [156]. However, this network is only able to incorporate undirected homogeneous graphs. As such, we modify the vertex feature update process to utilize per-edge-type per-direction message passing before the vertex feature reduction. The edge feature update process remains as it was in [157].

For each edge type ψ, we define two dense attention maps: γ_{ij}^{ψ} which indicates the importance of vertex v_j's features to vertex v_i when v_j is an outward neighbor of v_i over an edge of type ψ and ζ_{ij}^{ψ} which indicates the importance of vertex v_i's features to vertex v_j when v_i is an inward neighbor of v_j over an edge of type ψ. These two dense attention maps are formulated as

$$\gamma_{ij}^{\psi} = \frac{\sigma(e_{ij}^{\psi})}{\sum_{v_k \in \mathcal{O}^{\psi}(v_i)} \sigma(e_{ik}^{\psi}) + \epsilon} \tag{6.2}$$

$$\zeta_{ij}^{\psi} = \frac{\sigma(e_{ij}^{\psi})}{\sum_{v_k \in \mathcal{I}^{\psi}(v_j)} \sigma(e_{kj}^{\psi}) + \epsilon} \tag{6.3}$$

where e_{ij} denotes the edge feature vector input into the current layer associated with the edge from vertex v_i to vertex v_j, $\mathcal{O}^{\psi}(v_i)$ is the set of outward neighbors of vertex v_i over edges of type ψ, $\mathcal{I}^{\psi}(v_i)$ is the set of inward neighbors of vertex v_i over edges of type ψ, σ is the sigmoid function, and ϵ is a small fixed constant for numerical stability.

Utilizing these dense attention maps, we define our vertex feature update formula that as

$$x_i' = x_i + ReLU(BN(W_1 x_i + \sum_{\mathcal{E}_\psi \in \mathcal{E}} (\alpha^\psi(v_i) \ + \ \beta^\psi(v_i))) \tag{6.4}$$

$$\alpha^\psi(v_i) = \sum_{v_j \in \mathcal{O}^k(v_i)} \gamma_{ij}^\psi \odot W_2^\psi h_j \tag{6.5}$$

$$\beta^\psi(v_i) = \sum_{v_j \in \mathcal{I}^k(v_i)} \zeta_{ji}^\psi \odot W_3^\psi h_i \tag{6.6}$$

where h_i denotes the vertex feature vector input into the current layer associated with vertex v_i, h_i' denotes the vertex feature vector associated with vertex v_i output by the current layer, $\odot$ is the Hadamard product, ReLU is the rectified linear unit, and BN is batch normalization.

The edge feature update formula is computed as

$$e_{ij}^{\psi\prime} = e_{ij}^\psi + ReLU(BN(W_4^\psi e_{ij}^\psi + W_5 x_i + W_6 x_j)) \tag{6.7}$$

as in [157].

Multi-Layer Perceptron

The mutex edge feature vectors output by the last layer of the G-GCN are used to compute the probability (p_{ij}) of each task ordering (δ_{ij}) having a value of 1 or 0 in the optimal set of task orderings ρ^*. Each $p_{ij} \in [0, 1]$ is given by a Multi-Layer Perceptron (MLP):

$$p_{ij} = MLP(e_{v_{c_i} v_{s_j}}^{mux} || e_{v_{c_j} v_{s_i}}^{mux}), \quad \forall (i, j) \in \mathcal{M} \tag{6.8}$$

where $||$ is the concatenation operator.

Loss Function

For training the model, we assume the availability of a set of training data labeled with the optimal solution. Given the optimal set of task orderings ρ^*, we minimize the binary cross-entropy loss

$$L = -\frac{1}{|\mathcal{M}|} \sum_{(i,j)\in\mathcal{M}} \delta^*_{ij} log(p_{ij}) + (1 - \delta^*_{ij}) log(1 - p_{ij}) \tag{6.9}$$

averaged over mini-batches.

6.6 Heterogeneous Temporal Graph Scheduler

The Heterogeneous Temporal Graph Scheduler (HTGS) assumes a trained model and conducts a three-step process to solve the HCSD Problem. It first encodes the problem as a heterogeneous graph as described in Section 6.4. It then passes the heterogeneous graph to the trained model which outputs a set of the probabilities (p_{ij}) of each task ordering (δ_{ij}) having a value of 1 or 0 in the optimal set of task orderings ρ^*. Directly converting this output to a set of task orderings will generally yield invalid schedules due to temporal inconsistency. As such, we employ two possible search-based decoding strategies to convert this set of probabilities into a valid task ordering which we describe below.

6.6.1 Greedy Search

The greedy search strategy operates on a space of task orderings and starts from an initial search node where none of the task orderings have been set. For the initial node, we construct a Simple Temporal Network (STN) [33] from the precedence constraints, task durations, and deadlines.

At each expansion step, the greedy search selects the task ordering δ_{ij} with the highest directional probability, where directional probability is calculated as $|2 * (p_{ij} - 0.5)|$. It then sets the value of δ_{ij} to $\lfloor p_{ij} \rceil$ where $\lfloor \cdot \rceil$ is the nearest integer operator.

It is possible that setting a specific task ordering forces others task orderings to be set. As an example, say there is a precedence constraint $\tau_i \prec \tau_j$ and that there are mutex constraints $\tau_i \leftrightarrow \tau_k$ and $\tau_j \leftrightarrow \tau_k$. If we set the task ordering δ_{jk} to 1, then the task ordering δ_{ik} is forced to be 1 or there would be a temporal inconsistency where a task would have to occur before itself. While we have described this transitive property for the simple triangular case, it is possible that the property cascades and that both the setting of δ_{jk} and the setting of δ_{ik} can force more task orderings to be set. We update our STN with edges to represent these new transitive task orderings. This update can efficiently be completed in polynomial time [158].

Once the task orderings are set we need to check if any deadlines have been violated. This can be achieved through an All-Pairs Shortest Path (APSP) algorithm that can handle negative edges (e.g. Bellman-Ford [159]) on our STN to create a distance graph. If no negative cycles are detected in the distance graph then a child node is created with the updated set of task orderings and the updated STN, and the greedy search continues from the child node.

If a negative cycle is detected then the greedy search attempt to set the task ordering with the second highest directional probability. It continues down the list of possible task orderings according to directional probability until it is able to set one without any temporal inconsistencies or deadline violations and a child node is created.

A solution is found when all task orderings have been set. When all task orderings have been set, the distance graph used to check for temporal inconsistencies or deadline violations can easily be checked for the makespan of the solution. The set of task orderings ρ and the makespan C are output by our approach.

6.6.2 Beam Search

A beam search is a limited-width breadth-first search [160]. When using this search strategy, our approach operates on the same space as when using the greedy search and starts

from the same initial search node. At each level of the search tree, beam search generates all of children nodes from the nodes at the current level, however, all but the B best children according to the probability of the partial set of task orderings are pruned. The probability of the partial set of task orderings is defined as

$$p(\hat{\rho}) = \prod_{\hat{\delta_{ij}} \in \hat{\rho}} (\hat{\delta_{ij}} p_{ij} + (1 - \hat{\delta_{ij}})(1 - p_{ij})) \tag{6.10}$$

where $\hat{\rho} \subseteq \rho$ is a set of the individual task orderings whose values have been set.

The larger the value of B, which is known as beam width, the fewer search nodes are pruned, and with an infinite beam width the beam search is identical to breadth-first search. Children nodes are created using the same expansion process as that of the greedy search (i.e, using the transitive property to determine if additional task orderings should be set and then checking if deadlines have been violated).

For our approach, the beam search will have a maximum depth of $|\rho|$. Upon reaching this depth, instead of pruning all but the best B child nodes based on the probability of the partial set of task orderings we simply find the singular best child node based on makespan. The task ordering ρ and makespan C from this child node are output as the solution from our approach.

6.7 Experimental Evaluations

We evaluate the proposed approach via numerical experiments in a simulated emergency response domain used in prior work [60, 19, 63, 5]. In this domain, a diverse set of robots with different capabilities and speeds need to work together to rescue wounded survivors, deliver medicine to hospitals, put out fires, clear rubble, and rebuild damaged infrastructure, all while not exceeding any deadlines. For all of our experiments, a graph representation of the maps used in the Robocup Rescue Simulation League [60, 161] were used.

The experimental campaign consists of three parts. The first part aims at assessing the

performance of different variants of the HG-GCN presented in Section 6.5. The second part evaluates the proposed scheduling framework in its entirety through comparison against an optimal baseline and a state-of-the-art scheduling algorithm. The third party evaluates how HTGS improves the scalability of our heterogeneous multi-robot coordination framework GRSTAPS.

We implemented our approach in C++ with LibTorch [162]. In all of the experiments, we use the GUROBI MILP solver [163] with default parameters both for providing the optimal solution for training and as a baseline for evaluating the resulting performance of our approach. All experiments are run on a computer equipped with an AMD 3970X CPU, an A6000 GPU, and 32 GB of RAM.

6.7.1 Ablation Study

We randomly generated a training dataset of 200,000 HCSD problems from the emergency response domain by randomly sampling the number of robots, survivors, fires, and damaged buildings. Each problem had between 5-10 robots and 10-25 tasks, which would be in the small problem size as described below. We also randomized the locations of the survivors, fires, damaged buildings, hospitals, and the robots' initial locations. Deadlines for putting out fires and transporting survivors to hospitals were set based on the equations from the Robocup Rescue Simulator [161]. The specific set of tasks and the allocation of robots to tasks were predetermined for each of these training problems.

We trained different variants of the HG-GCN architecture outlined in Section 6.5 by considering different combinations of: number of HG-GC layers (4, 8, 26) and number of training samples (1k, 5k, 10k). All models used 6 layers for our Multi-Layer Perceptron. The feature dimension for each hidden layer was 64 and we used the Adam optimizer [164] with a learning rate of $1e-3$. The batch size for training was 10.

We then evaluated the F1 score for each of these variants on 1000 randomly generated problems for each of three sizes:

- Small: [5, 15] robots and [10, 25] tasks
- Medium: [16, 30] robots and [26, 50] tasks
- Large: [31, 45] robots and [51, 75] tasks

Table 6.1: Ablation Study of the F1 score for different variants of the HG-GCN architecture. Red: The best model. Violet: The second best model.

# Layers	# Samples	Small	Medium	Large
4	10k	0.689 ± 0.140	0.602 ± 0.104	0.538 ± 0.063
	50k	0.715 ± 0.135	0.634 ± 0.105	0.581 ± 0.065
	100k	0.732 ± 0.104	0.666 ± 0.105	0.628 ± 0.071
8	10k	0.644 ± 0.172	0.613 ± 0.128	0.587 ± 0.101
	50k	0.759 ± 0.124	0.675 ± 0.100	0.657 ± 0.063
	100k	0.799 ± 0.063	0.729 ± 0.079	0.673 ± 0.055
16	10k	0.749 ± 0.167	0.701 ± 0.184	0.653 ± 0.176
	50k	0.871 ± 0.133	0.842 ± 0.154	0.783 ± 0.176
	100k	0.993 ± 0.074	0.975 ± 0.064	0.907 ± 0.093

In general we observe in Table 6.1, as expected, that increasing the number of layers and number of training samples results in better performance. We select the model that obtains the best F1 results for the remaining experiments.

6.7.2 Comparison with other Scheduling Approaches

For our second experiment, we benchmarked our approach with both search strategies against solving the MILP formulation from Equation 6.1 with GUROBI and the MILP-based scheduling algorithm used by DITAGS in [153] which we will call $DITAGS_{sched}$. For $DITAGS_{sched}$, we add temporal deadlines to their formulation as the two equations 6.1g and 6.1h. We refer to the variant of our approach that utilizes the greedy search as HTGS_{greedy} and the variant of our approach that utilizes the beam search as HTGS_{beam}.

We used a beam width of 10. Each approach was given a 30s timeout for each problem instance.

We compared all four approaches on problem coverage, optimality gap, solution makespan (C), and computation time. For both solution makespan and computation time, we computed separate means and standard deviations for problems where an optimal solution could be computed within the time limit and problems where it could not. For each problem size, we randomly generated 1000 HCSD Problems to evaluate with.

Table 6.2: Summary of benchmarking results for HCSD Problems. Red: The best value for each metric. Violet: The second best value for each metric.

		Approaches			
		MILP	DITAGS$_{sched}$	HTGS$_{greedy}$	HTGS$_{beam}$
Small	Cov. (%)	100	100	100	100
	Opt. gap (%)	0	0.8 ± 1.0	3.7 ± 1.9	2.1 ± 0.9
	C - opt.	$2,565.8 \pm 306.8$	$2,586.0 \pm 301.0$	$2,660.8 \pm 321.7$	$2,620.0 \pm 314.1$
	Comp. Time - opt. (s)	0.041 ± 0.028	0.010 ± 0.003	0.019 ± 0.007	0.021 ± 0.014
Medium	Cov. (%)	68.9	100	100	100
	Opt. gap (%)	0	5.9 ± 5.1	8.0 ± 4.9	4.1 ± 2.9
	C - opt.	$3,656.6 \pm 421.1$	$3,886.3 \pm 576.1$	$3,949.5 \pm 488.6$	$3,805.8 \pm 451.2$
	C - no opt.	N/A	$5,526.6 \pm 142.6$	$5,750.6 \pm 661.9$	$5,392.2 \pm 466.3$
	Comp. Time - opt. (s)	10.277 ± 4.961	0.096 ± 0.022	0.043 ± 0.021	0.060 ± 0.034
	Comp. Time - no opt. (s)	N/A	0.566 ± 0.362	0.051 ± 0.025	0.093 ± 0.031
Large	Cov. (%)	0	74.6	100	92.3
	C - no opt.[a]	N/A	$6,980.6 \pm 725.1$	$7,662.5 \pm 890.5$	$6,771.3 \pm 628.4$
	Comp. Time - no opt. (s) [a]	N/A	19.723 ± 13.287	0.929 ± 0.453	1.987 ± 1.456

[a]Computed using problems for which all three satisficing approaches were able to solve the problem.

MILP

As can be seen in Table 6.2, MILP was always the slowest approach when it was able to compute a solution. This is because while MILP is using the commercially developed solver in GUROBI, it is the only approach that must prove that its solutions are optimal. Furthermore, it is unable to solve all of the medium problems in the time limit - it solved 68.9% - and was unable to solve any of the large problems in the time limit. This is due to the combinatoric nature of the problem - as the number of tasks and the number of robots

increase, the number of mutex constraints increases combinatorically. The combinatorics become further exaggerated as the problem which is an instance of a Disjunctive Temporal Problem [137] which is NP-Hard with respect to the number of disjunctions (i.e., the number of mutex constraints). This can be seen as MILP on average takes about 250x as long to solve a medium problem as a small problem.

$DITAGS_{sched}$

For small problems, $DITAGS_{sched}$ has an average optimality gap of less than 1% and, on average, is the fastest approach. For medium and large problems, the combinatoric factors described for MILP start to affect the results, but not to the same degree as for MILP. This is because $DITAGS_{sched}$ also solves a MILP as part of its approach. However, unlike MILP, $DITAGS_{sched}$ is a satisficing approach that attempts to find quality solutions but is not concerned with discovering the optimal solution. As such, it can solve all of the medium problems but only 74.6% of the large problems in the time limit. Additionally, for medium and large problems, it is, on average, the slowest of the 3 satisficing approaches, and for large problems is an order of magnitude slower than either of the learned approaches but has the second-best solution quality on average.

HTGS$_{greedy}$

For small problems, HTGS$_{greedy}$ is the second fastest approach behind only $DITAGS_{sched}$. This is likely due to the $DITAGS_{sched}$ using a commercially developed solver and computational overhead from the deep architecture used by HTGS$_{greedy}$. For medium and large problems, HTGS$_{greedy}$ is the fastest approach and is the only approach to solve all of the large problems. For medium and large problems, it is on average two orders of magnitude faster than MILP on problems that could be solved optimally, an order of magnitude faster than $DITAGS_{sched}$, and takes about half of the time of HTGS$_{beam}$. This is speed is caused by a greedy search that only expands at most $|\mathcal{M}|$ nodes and only visits at most $\frac{|\mathcal{M}|^2}{2}$ nodes

making both the number of expansions and the number of visits polynomial with the number of mutex constraints. This combined with incremental updates to the STN and distance matrix during search results in an efficient approach. The tradeoff for this increase in efficiency is HTGS$_{greedy}$ has the lowest quality solutions on average, but for problems where an optimal solution could be computed, it still, on average, has an optimality gap less than 10% and for problems where an optimal solution could not be computed HTGS$_{greedy}$ computed solutions that were on average less than 10% worse than the best solution that the other approaches could find. This demonstrates a very efficient approach that still produces comparable solution quality.

HTGS$_{beam}$

For small problems, HTGS$_{beam}$ is marginally slower than HTGS$_{greedy}$, but is still about twice as fast as MILP. For medium and large problems that HTGS$_{beam}$ could solve (it only solved 92.3% of the large problems), it on average, had the highest quality solutions of the satisficing approaches. This is likely a result of it using the beam width to explore more of the search tree than the greedy search and using the makespan to select the solution during the final expansion. Additionally, for these problems, it was the second fastest approach only behind HTGS$_{greedy}$ and was about an order of magnitude faster than $DITAGS_{sched}$. This speed is caused by the beam search only expanding at most $(|\mathcal{M}| - 1) * B + 1$ nodes and visiting at most $B|\mathcal{M}|^2 + (2B + 1)|\mathcal{M}| - B$ nodes making both the number of expansions and the number of visits polynomial with the number of mutex constraints. The beam search is sped up further through a parallelized implementation. However, even with the parallelized implementation, there is additional overhead for each expansion when compared to the greedy search in HTGS$_{greedy}$. This demonstrates an efficient approach that trades some speed for improvement in solution quality.

6.7.3 Usage within the GRSTAPS framework

For our third experiment, we demonstrated how the efficiency of HTGS aids our heterogeneous multi-robot coordination framework GRSTAPS in solving larger problems with more robots and tasks within a specified amount of time. For this experiment, we generate 100 large Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) problems. In these problems, the specific set of tasks and allocation of robots to tasks were not predetermined for these problem instances.

GRSTAPS utilizes a scheduling algorithm to make decisions about what tasks to include in the plan and what robots to allocate to each task. As part of solving the problem, the scheduling algorithm is run thousands of times on problems of various sizes as GRSTAPS searches for the best tasks and task allocations. In this experiment, we refer to each variant of GRSTAPS by the scheduling algorithm it uses. As the optimal baseline was not able to solve any of the large problems in the previous experiment, we did not use it for this experiment. Each approach was given a 10-minute timeout and was evaluated on problem coverage, solution makespan (C), and computation time. For solution makespan and computation time, we compute separate means and standard deviations for problems that all three approaches were able to solve and for problems that only both of the HTGS approaches were able to solve.

$DITAGS_{sched}$

As we observe in Table 6.3, when GRSTAPS uses $DITAGS_{sched}$ as its scheduling algorithm, it is only able to solve 3 problems. This is caused by the large amount of time needed to solve large scheduling problems. This extra time compounds and significantly increases the time that GRSTAPS needs to solve a problem. As with the previous experiment, when $DITAGS_{sched}$ is able to solve a problem, it tends to have the second best solution quality of the satisficing approaches behind HTGS$_{beam}$.

	Approaches		
	DITAGS$_{sched}$	HTGS$_{greedy}$	HTGS$_{beam}$
Cov. (%)	3	68	33
C_{ALL} [a]	$6,477.6 \pm 441.0$	$7,180.1 \pm 499.5$	$6,234.3 \pm 333.4$
Comp. Time$_{ALL}$ (s)[a]	541.0 ± 113.5	356.1 ± 23.83	431 ± 54.7
C_{HTGS} [b]	N/A	$7,654.2 \pm 503.0$	$6,771.6 \pm 343.6$
Comp. Time$_{HTGS}$ (s)[b]	N/A	411.3 ± 34.8	511.8 ± 123.4

Table 6.3: Summary of benchmarking results for STAP-STC Problems. Red: The best value for each metric. Violet: The second best value for each metric.

[a] Computed using all problems for which all three approaches were able to solve the problem.
[b] Computed using all problems for which both HTGS$_{greedy}$ and HTGS$_{beam}$ could solve.

HTGS$_{greedy}$

In comparison to the other two approaches, we observe that when GRSTAPS uses HTGS$_{greedy}$ as its scheduling algorithm, it is able to solve 68 problems which is more than twice as many as HTGS$_{beam}$ and more than an order of magnitude more than $DITAGS_{sched}$. This is caused by the efficiency of HTGS$_{greedy}$ as it is the only approach to average less than 1s to solve large scheduling problems. When HTGS$_{greedy}$ is used by GRSTAPS this speed compounds as the scheduling algorithm is run thousands of times. This speed comes at the expense of having a worse solution quality than the other two approaches for the subset of problems that they can solve; however, on average, it produced solutions with makespans within 15% of those produced by the other approaches. This demonstrates that HTGS$_{greedy}$ aids GRSTAPS in being able to solve significantly more large problems than prior work through being a very efficient approach that produces solutions with similar quality to state-of-the-art.

HTGS$_{beam}$

When GRSTAPS uses HTGS$_{beam}$ as its scheduling algorithm, we observe that it is able to solve 33 of the problems. This is less than half of the number of problems that HTGS$_{greedy}$

can solve it is still significantly more than the prior work of $DITAGS_{sched}$. Additionally, for the problems that it can solve, it produces the highest quality schedules of any of the approaches. This is likely a result of it using the beam width to explore more of the search tree than the greedy search and using the makespan to select the solution during the final expansion. This demonstrates that HTGS$_{greedy}$ aids GRSTAPS in being able to solve more large problems than prior work through being an efficient approach that on average, produces superior quality solutions than state-of-the-art.

6.8 Discussion & Conclusion

In this chapter, we introduced a novel learning-based approach named Heterogeneous Temporal Graph Scheduler (HTGS) to solve the Heterogeneous Coalition Scheduling with Deadlines (HCSD) Problem. This approach utilizes a Heterogeneous Graph Convolution Network-based model which outputs the probability that each individual task ordering is in the optimal solution and a search-based method that utilizes these probabilities to set task orderings. We first present an ablation study where we determine the number of GNN layers and the number of training samples to use for our model. We then demonstrate that our approach is more efficient than an optimal baseline and a state-of-the-art scheduling algorithm while still generating comparable solutions. Furthermore, we demonstrate that the proposed approach scales to larger problems. Finally, we demonstrate how HTGS aids in guiding a higher-level multi-robot coordination framework in GRSTAPS to high-quality solutions for large problems.

However, HTGS does have its limitations and there are open questions left to explore. First, it assumes quality data is available to train with however previous literature tends to think this is an acceptable assumption [157, 165]. Second, while all 13 of Allen's temporal relationships can be represented in the heterogeneous graph that we construct, further research can be conducted to explore how well this method handles each different type of temporal relationship. Third, recent research in exploring autoregressive approaches has

demonstrated better results in terms of generalizing from small to large problems [157, 166]. Future work can explore updating HTGS as an autoregressive approach and other methods that can improve the generalization capabilities.

CHAPTER 7
CONCLUSION AND FUTURE DIRECTIONS

Throughout this dissertation, we have made several contributions to the field of heterogeneous multi-robot coordination. We focused on representations and frameworks that allow independent algorithms to focus on the properties of tasks and robots from their individual sub-problems, while leveraging information discovered by other algorithms utilizing different properties, to simultaneously solve a large-scale heterogeneous multi-robot coordination problem. Specifically, this thesis **examines the use of shared constraints on tasks and robots to interleave algorithms for task planning, task allocation, scheduling, and motion planning and investigates the hypothesis that a framework that interleaves algorithms to these four sub-problems will lead to solutions with lower makespans, greater computational efficiency, and the ability to solve larger problems**.

7.1 Summary of Contributions

This dissertation has made the following contributions to validate this claim:

7.1.1 Interleaving Task Planning and Scheduling

We contributed Forward Chaining Partial-Order Planner (FCPOP), which interleaves task planning and scheduling layers to solve the temporal planning problem. FCPOP leverages techniques from state-space-based forward chaining for efficiency and more informed search guidance and techniques from plan-space-based partial-order planning for flexibility. Furthermore, we contributed Forward Chaining Hierarchical Partial-Order Planner (FCHPOP), a hierarchical temporal planner, which builds upon FCPOP through the integration of techniques from Hierarchical Task Networkss. FCHPOP utilizes and refines abstract tasks to benefit from domain expert knowledge and reduce the number of search

steps needed to discover a solution. We demonstrated that FCPOP and FCHPOP outperformed state-of-the-art baselines on common benchmark problems.

7.1.2 Interleaving Task Allocation, Scheduling, and Motion Planning

We contributed Incremental Task Allocation Graph Search (ITAGS), which interleaves task allocation, scheduling, and motion planning layers to solve the trait-based time-extended task allocation problem. ITAGS is a search-based approach that leverages two novel complementary heuristics in Allocation Percentage Remaining (APR) and Normalized Schedule Quality (NSQ) and a convex combination of the two heuristics in Time-Extended Task Allocation Quality (TETAQ). Our experiments demonstrate the relative influence of each of these heuristics and illustrate the benefits of interleaving information between layers through shared constraints. Furthermore, we show that ITAGS outperformed state-of-the-art baselines on benchmark problems.

7.1.3 Interleaving Allocation, Planning, and Scheduling

We contributed the formulation of a novel holistic heterogeneous multi-robot coordination problem named Simultaneous Task Allocation and Planning with Spatiotemporal Constraints (STAP-STC) that simultaneous considers all four questions of coordination: *what* (task planning), *who* (task allocation), *when* (scheduling), *how* (motion planning). Furthermore, we contribute an initial solution to the STAP-STC problem named Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling (GRSTAPS). GRSTAPS builds upon our previous interleaved frameworks, FCPOP and ITAGS, to interleave layers for all four sub-problems. Our experiments demonstrate the benefits of interleaving information through shared constraints over two sequentially chained baselines. Furthermore, we demonstrate the benefits of this approach over trying to solve a monolithic version of the problem through a comparison of GRSTAPS and three state-of-the-art temporal planners that are given a discretized version of the problem.

7.1.4 Heterogeneous Coalition Scheduling with Temporal Uncertainty

We contributed the formulation of a novel uncertainty-aware scheduling problem named Heterogeneous Coalition Scheduling with Temporal Uncertainty (HCSTU) as a subset of the STAP-STC problem. The HCSTU explicitly considers the uncertainties in the durations of tasks and task transitions. Additionally, we contribute a sampling-based risk-aware approach named Coalition Scheduling with Heuristic Sample Selection and Risk Guarantee (CS-HSSRG) that provides a a theoretical guarantee on a user provided acceptable amount of risk. We demonstrate that CS-HSSRG is more efficiency that state-of-the-art approaches, while not being overly conservative, and that CS-HSSRG aids GRSTAPS in making robust decisions about what tasks to include in the task plan and what robots to allocate to those tasks.

7.1.5 Learning to Set Task Orderings for Heterogeneous Coalition Scheduling with Deadlines

We contribute a Heterogeneous Gated Graph Convolution Network (HG-GCN) model that learns the probability that pairs of tasks are ordered in a specific way in an optimal schedule. Furthermore, we contribute a search-based approach named Heterogeneous Temporal Graph Scheduler (HTGS) with two search strategies that utilizes the output of the HG-GCN model to approximately solve the Heterogeneous Coalition Scheduling with Deadlines problem. We demonstrate the efficiency of this approach against state-of-the-art baselines and the compounded benefits of utilizing it within GRSTAPS.

7.2 Open Questions

While the contributions of this dissertation have explored the benefits of interleaving information through shared constraints within the context of the multi-robot coordination problem, there remain many open questions at various levels of scope for the problem considered during this dissertation. The following subsections provide open questions at

varying levels of scope for this problem.

7.2.1 Reducing Motion Planning Assumptions

Due to the complexity of the STAP-STC problem, we made a number of assumptions so that we could create our initial solution, GRSTAPS. Several of these assumptions affect the motion planning layer leaving ample room to improve both the motion planning layer itself and its interaction with the other layers.

First, the motion planning layer current does not consider collisions between robots and makes the assumption that modern motion controllers can handle avoiding collisions with moved objects [82, 87, 100]. For the domains considered in this dissertation, tight coupling of robot motion was not required (there were no instances of cooperative manipulation, and no narrow passageways that could lead to deadlock). For domains where more intricate robot-to-robot interaction is required (e.g., multiple mobile manipulators cooperating to perform household tasks), it will be essential to include robot interaction when planning collision-free paths. More advanced Multi-Agent Pathfinding (MAPF) [25, 116] or Multi-Agent Motion Planning (MAMP) [111, 167] algorithms may be more suitable for teams of heterogeneous robots, while for shared manipulation, some combination of sampling-based planning in composite configuration spaces and low-level cooperative control (during execution) may prove effective.

Second, we assume a simplistic, static representation of the environment that does not update robots' free configuration spaces when robots rearrange objects in the work space. Again, because the domains considered in this dissertation did not require close interaction between robots and objects in the environment, this limitation was not problematic. It is possible to relax this assumption to allow for representation of more domains by utilizing techniques developed in the Task and Motion Planning community, in particular those developed for Multi-Modal Motion Planning [168, 169, 91]. Furthermore, through commonly used representations, such as the Kinematic Graph [168], a connection can be made

between the symbolic objects used by task planning and the geometric objects used by motion planning. These connections can be utilized to make shared constraints on the parameters of grounded tasks to prune operators or partially grounded operators which will reduce the overall branching factor of task planning.

Third, currently the motion planning layer generates a negative constraint for a motion planning query if it determines that creating a motion plan is infeasible or if it times out. When a sampling-based motion planning algorithm is used that is probabilistically complete, then GRSTAPS is not complete as the infeasibility of a motion plan cannot be determined. Recent and ongoing research is exploring general approaches for constructing proofs of infeasibility for sampling-based motion planning algorithms [66, 67, 68]. These approaches could be incorporated to make GRSTAPS a complete algorithm regardless of motion planning algorithm. Furthermore, they would improve the efficiency of GRSTAPS as it would not need to spend time computing a motion plan when one is infeasible.

7.2.2 Uncertainty

While we pursued an initial exploration into uncertainty in Chapter 5, there are numerous sources of uncertainty within the context of the STAP-STC problem and numerous approaches that can be developed to improve robustness within the context of an interleaved framework like GRSTAPS. We discuss a few of them here.

First, there is the possibility for uncertainty in the effects of the symbolic tasks. There has been significant research on probabilistic planning for a single agent focusing on Markov Decision Process-based approaches [170, 171, 172]. These probabilistic planners do not reason about temporal constraints and are ill-suited for the multi-robot planning problem as the size of the search space increases exponentially with the number of robots [173], however, it is possible that there are techniques that can be utilized or modified to work within the context of FCPOP or GRSTAPS. Incorporating probabilistic planning techniques for individual robots as an additional layer underneath the task allocation layer, similar to ap-

proach used by [173], is potentially feasible, however, further analysis would have have to be conducted on the scalability with respect to the number of robots on such an approach.

Second, there is the possibility for uncertain robot traits or task trait requirements. In our current formulation, we assume that each robot of the same species has the same traits, but in a real scenario, a robot's trait could be different from others of the same species due to different conditions (e.g., wear and tear, battery life, sensor failure, etc). Furthermore, knowing the exact trait requirements to execute a task is not always viable. Past research has considered this type of uncertainty for an instantaneous task allocation problem [41, 174], however, to the best of our knowledge there are no approaches that consider trait uncertainty for the time-extended problem. It is possible to build upon APR through incorporating the Sequential Probability Ratio Test that we used to reason about temporal uncertainty in Chapter 5. This would allow for a user-specified risk tolerance on making sure that each coalition had sufficient traits for each task it was assigned.

Third, there is the possibility that traversibility is uncertain. There are many real world scenarios where information about the environment is uncertain (e.g., stale map information, weather causing potential road blockages, etc). Modern approaches to solving this type of problem, including the Canadian Traveler Problem (CTP) that we mentioned in Chapter 5, are online approaches that utilize information discovered as the robot traverses the environment, however, there may be techniques from these approaches that could be incorporated into a framework like GRSTAPS.

Finally, each of these types of uncertainty has ramifications on other layers than than the one that we mentioned in its paragraph above. For instance, if a road is blocked a robot could take significantly more time to transition to a task. This may in turn make another robot that is further way, but has a more direct route, better suited to participate in the task. Additionally, uncertainties might stem from the integration of sub-problems. For example, for task that is selected the outcome probabilities may depend on the coalition assigned to it. Furthermore, having contingencies might be needed to decrease the risk of failure.

7.2.3 Execution

Finally, GRSTAPS is an offline algorithm that assumes the solution it generates can be executed perfectly, however, in real problems, there are numerous opportunities for execution to fail. The obvious solution for execution failure would be to re-solve the problem from the new initial state, however, this would be time consuming and fails to utilize the large amount of information contained in the previous solution, the previous searches, and the previous shared constraints. It is possible that the new solution may only need a small deviation from the current solution to still be a viable solution. This opens up several questions on how to use the information/modules in GRSTAPS to best repair a solution. Is it better to try to repair the solution through a monolithic approach or interleave algorithms such as those found in [175, 153] in a similar manner to how GRSTAPS interleaves its current modules? How can the current information discovered such as open/closed/pruned task plan nodes, open/closed/pruned task allocation nodes be used to improve/speed up the repair? Should an algorithm select what portion of the solution to repair based on the type of execution failure or another metric? If execution changes beyond what is expect, but execution does not fail should the algorithm still attempt to repair the solution or just allow it to continue to execute?

REFERENCES

[1] J. Liu and R. K. Williams, "Coupled temporal and spatial environment monitoring for multi-agent teams in precision farming," *IEEE Conference on Control Technology and Applications*, pp. 273–278, 2020.

[2] S. Jeon, J. Lee, and J. Kim, "Multi-Robot Task Allocation for Real-Time Hospital Logistics," in *International Conference on Systems, Man, and Cybernetics*, 2017.

[3] C. J. McCook and J. M. Esposito, "Flocking for heterogeneous robot swarms: A military convoy scenario," in *Proceedings of the Annual Southeastern Symposium on System Theory*, 2007, pp. 26–31.

[4] R. Stern *et al.*, "Multi-agent pathfinding: Definitions, variants, and benchmarks," in *International Symposium on Combinatorial Search*, 2019, pp. 151–158.

[5] G. Neville, A. Messing, H. Ravichandar, S. Hutchinson, and S. Chernova, "An Interleaved Approach to Trait-Based Task Allocation and Scheduling," in *International Conference on Intellifent Robots and Systems*, IEEE, 2021.

[6] A. Torreño, E. V. A. Onaindia, and U. P. D. València, "Cooperative Multi-Agent Planning : A Survey," *ACM Computing Surveys*, vol. 50, no. 6, pp. 1–32, 2017.

[7] S. Irnich, P. Toth, and D. Vigo, "The Family of Vehicle Routing Problems," in *Vehicle Routing*, 2014, ch. 1, pp. 1–33.

[8] R. Shome and K. E. Bekris, "Anytime multi-arm task and motion planning for pick-and-place of individual objects via handoffs," in *International Symposium on Multi-Robot and Multi-Agent Systems*, 2019.

[9] D. Bredström and M. Rönnqvist, "Combined vehicle routing and scheduling with temporal precedence and synchronization constraints," *European Journal of Operational Research*, vol. 191, no. 1, pp. 19–31, 2008.

[10] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "An incremental constraint-based framework for task and motion planning," *International Journal of Robotics Research*, vol. 37, no. 10, pp. 1134–1151, 2018.

[11] D. G. MacHaret and M. F. Campos, "A survey on routing problems and robotic systems," *Robotica*, vol. 36, no. 12, pp. 1781–1803, 2018.

[12] C. R. Garrett, "Sampling-Based Task and Motion Planning for Robots in the Real World," Ph.D. dissertation, 2021.

[13] A. Wald, "Sequential Tests of Statistical Hypotheses," *The Annals of Mathematical Statistics*, vol. 16, no. 2, pp. 117–186, 1945.

[14] J. Allen, "Maintaining knowledge about temporal intervals," *Communications of the ACM*, vol. 26, no. 11, pp. 832–843, 1983.

[15] O. Sapena, A. Torrenõ, and E. Onaindiá, "Parallel heuristic search in forward partial-order planning," *Knowledge Engineering Review*, vol. 31, no. 5, pp. 417–428, 2016.

[16] J. Benton, A. Coles, and A. Coles, "Temporal planning with preferences and time-dependent continuous costs," in *International Conference on Automated Planning and Scheduling*, 2012.

[17] J. Kvarnström, "Planning for loosely coupled agents using partial order forward-chaining," in *International Conference on Automated Planning and Scheduling*, 2011.

[18] A. Coles, A. Coles, M. Fox, and D. Long, "Forward-Chaining Partial-Order Planning," in *International Conference on Automated Planning and Scheduling*, 2010.

[19] P. Bechon, M. Barbier, G. Infantes, C. Lesire, and V. Vidal, "HiPOP: Hierarchical Partial-Order Planning," *STAIRS*, pp. 51–60, 2014.

[20] A. Coles, A. Coles, M. Martinez, and P. Sidiropoulos, *International Planning Competition 2018 Temporal Track*.

[21] R. Lallement, L. De Silva, and R. Alami, "HATP: An HTN Planner for Robotics," in *Workshop on Planning and Robotics (PlanRob)*, 2014.

[22] S. Edelkamp and J. Hoffmann, "PDDL2.2: The Language for the Classical Part of the 4th International Planning Competition," Tech. Rep., 2004.

[23] *International Planning Competition 2011 Temporal Track*, 2011.

[24] J. Rosell, "Assembly and task planning using Petri nets: A survey," *Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture*, vol. 218, no. 8, pp. 987–994, 2004.

[25] G. Sharon, R. Stern, A. Felner, and N. R. Sturtevant, "Conflict-based search for optimal multi-agent pathfinding," *Artificial Intelligence*, vol. 219, pp. 40–66, 2015.

[26] Y. H. Kim and B. K. Kim, "A multi-robot task planning system minimizing the total execution time for hospital service," in *International Conference on Control, Automation and Systems*, 2010.

[27] C. Galindo, J. A. Fernández-Madrigal, J. González, and A. Saffiotti, "Robot task planning using semantic maps," *Robotics and Autonomous Systems*, vol. 56, no. 11, pp. 955–966, 2008.

[28] B. Schattenberg, "Hybrid Planning and Scheduling," Ph.D. dissertation, 2009.

[29] G. J. Sussman, "A computational model of skill acquisition," Ph.D. dissertation, 1973.

[30] D. S. Weld, "An Introduction to Least Commitment Planning," *AI magazine*, vol. 15, no. 4, pp. 27–27, 1994.

[31] Q. Yang, *Intelligent planning: a decomposition and abstraction based approach*. Springer Science & Business Media, 2012.

[32] H. L. Younes and R. G. Simmons, "VHPOP: Versatile heuristic partial order planner," *Journal of Artificial Intelligence Research*, vol. 20, pp. 405–430, 2003.

[33] R. Dechter, I. Meiri, and J. Pearl, "Temporal constraint networks," *Artificial Intelligence*, vol. 49, pp. 61–95, 1991.

[34] A. Bit-Monnot, "A constraint-based encoding for domain-independent temporal planning," in *International Conference on Principles and Practice of Constraint Programming*, 2018.

[35] P. Eyerich, R. Mattmüller, and G. Röger, "Using the Context-Enhanced Additive Heuristic for Temporal and Numeric Planning," in *International Comference on Planning and Scheduling*, 2009.

[36] M. Helmert, "The fast downward planning system," *Journal of Artificial Intelligence Research*, vol. 26, pp. 191–246, 2006.

[37] O. Sapena, E. Onaindia, and A. Torreño, "FLAP: Applying Least-Commitment in Forward-Chaining Planning," *AI Communications*, vol. 28, no. 1, pp. 5–20, 2015.

[38] J. Hoffman and B. Nebel, "The FF Planning System: Fast Plan Generation Through Heuristic Search," *Journal of Artifiial Intelligence (JAIR)*, pp. 253–302, 2001.

[39] O. Sapena and E. Onaindia, "TFLAP - Planner Abstract," in *International Planning Competition 2018 Temporal Track*, 2018.

[40] A. Prorok, M. A. Hsieh, and V. Kumar, "The Impact of Diversity on Optimal Control Policies for Heterogeneous Robot Swarms," *IEEE Transactions on Robotics*, 2017.

[41] H. Ravichandar, K. Shaw, and S. Chernova, "STRATA: A Unified Framework for Task Assignments in Large Teams of Heterogeneous Agents," *Journal of Autonomous Agents and Multi-Agent Systems*, 2019.

[42] B. P. Gerkey and M. J. Matarić, "A formal analysis and taxonomy of task allocation in multi-robot systems," *International Journal of Robotics Research*, vol. 23, no. 9, 2004.

[43] G. Korsah, A. Stentz, and M. Dias, "A comprehensive taxonomy for multi-robot task allocation," *International Journal of Robotics Research*, vol. 32, pp. 1495–1512, 2013.

[44] E. Nunes, M. Manner, H. Mitiche, and M. Gini, "A taxonomy for task allocation problems with temporal and ordering constraints," *Robotics and Autonomous Systems*, vol. 90, 2017.

[45] S. Sariel and T. Balch, "Dynamic and distributed allocation of resource constrained project tasks to robots," in *International Workshop on Multi-Agent Robotic Systems*, 2006.

[46] E. G. Jones, M. B. Dias, and A. Stentz, "Time-extended multi-robot coordination for domains with intra-path constraints," *Autonomous Robots*, vol. 30, pp. 41–56, 2011.

[47] S. Giordani, M. Lujak, and F. Martinelli, "A distributed multi-agent production planning and scheduling framework for mobile robots," *Computers and Industrial Engineering*, vol. 64, no. 1, 2013.

[48] M. Krizmancic, B. Arbanas, T. Petrovic, F. Petric, and S. Bogdan, "Cooperative Aerial-Ground Multi-Robot System for Automated Construction Tasks," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 798–805, 2020.

[49] M. F. Tompkins, "Optimization Techniques for Task Allocation and Scheduling in Distributed Multi-Agent Operations," Ph.D. dissertation, MIT, Cambridge MA, 2003.

[50] G. A. Korsah, B. Kannan, B. Browning, A. Stentz, and M. B. Dias, "xBots: An approach to generating and executing optimal multi-robot plans with cross-schedule dependencies," in *International Conference on Robotics and Automation*, IEEE, 2012, pp. 115–122.

[51] J. Guerrero, G. Oliver, and O. Valero, "Multi-robot coalitions formation with deadlines: Complexity analysis and solutions," *PLoS ONE*, vol. 12, no. 1, 2017.

[52] S. Thakar *et al.*, "Task assignment and motion planning for bi-manual mobile manipulation," in *International Conference on Automation Science and Engineering*, 2019.

[53] A. M. Kabir *et al.*, "Incorporating Motion Planning Feasibility Considerations during Task-Agent Assignment to Perform Complex Tasks Using Mobile Manipulators," in *International Conference on Robotics and Automation*, 2020.

[54] S. D. Ramchurn, M. Polukarov, A. Farinelli, C. Truong, and N. R. Jennings, "Coalition formation with spatial and temporal constraints," in *International Joint Conference on Autonomous Agents and Multiagent Systems*, vol. 2, 2010.

[55] L. Capezzuto, D. Tarapore, and S. D. Ramchurn, "Anytime and Efficient Coalition Formation with Spatial and Temporal Constraints," in *European Conference on Multi-Agent Systems*, 2020.

[56] M. Koes, I. Nourbakhsh, and K. Sycara, "Heterogeneous multirobot coordination with spatial and temporal constraints," *AAAI*, vol. 5, pp. 1292–1297, 2005.

[57] H. Choset *et al.*, *Principles of Robot Motion*. 2005.

[58] M. Gini, "Multi-robot allocation of tasks with temporal and ordering constraints," *AAAI*, pp. 4863–4869, 2017.

[59] F. Glover and M. Laguna, "Tabu Search," in *Modern Heuristic Techniques for Combinatorial Problems*, 1993.

[60] H. Kitano *et al.*, "RoboCup rescue: search and rescue in large-scale disasters as a domain for autonomous agents research," *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics*, vol. 6, pp. 739–743, 1999.

[61] A. Whitbrook, Q. Meng, and P. W. Chung, "A novel distributed scheduling algorithm for time-critical multi-agent systems," in *IEEE International Conference on Intelligent Robots and Systems*, vol. 2015-Decem, 2015, pp. 6451–6458.

[62] W. Zhao, Q. Meng, and P. W. Chung, "A Heuristic Distributed Task Allocation Method (PIA)," *IEEE Transactions on Cybernetics*, vol. 46, no. 4, pp. 902–915, Apr. 2016.

[63] A. Messing and S. Hutchinson, "Forward Chaining Hierarchical Partial-Order Planning," *International Workshop on the Algorithmic Foundations of Robotics*, vol. 14, 2020.

[64] R. Bohlin and L. E. Kavraki, "Path Planning Using Lazy PRM," *International Conference on Robotics and Automation (ICRA)*, no. April, 2000.

[65] I. Sucan, M. Moll, and L. E. Kavraki, "The Open Motion Planning Library (OMPL)," *IEEE Robotics & Automation Magazine*, vol. 19, no. 4, pp. 72–82, 2012.

[66] S. Li and N. T. Dantam, "Towards general infeasibility proofs in motion planning," in *International Conference on Intelligent Robots and Systems*, 2020.

[67] S. Li and N. Dantam, "Learning Proofs of Motion Planning Infeasibility," in *Robotics: Science and Systems*, 2021.

[68] S. Li and N. T. Dantam, "Exponential Convergence of Infeasibility Proofs for Kinematic Motion Planning," *International Workshop on the Algorithmic Foundations of Robotics*, pp. 1–16, 2022.

[69] E. Marie, E. H. Durfee, C. L. Ortiz, and M. J. Wolverton, "A Survey of Research in Continual Planning," *AI Magazine*, vol. 20, no. 4, pp. 13–22, 1999.

[70] F. Lagriffoul, N. T. Dantam, C. Garrett, A. Akbari, S. Srivastava, and L. E. Kavraki, "Platform-Independent Benchmarks for Task and Motion Planning," *IEEE Robotics and Automation Letters*, vol. 3, no. 4, pp. 3765–3772, 2018.

[71] C. Garrett *et al.*, "Integrated Task and Motion Planning," *International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1–30, 2020.

[72] T. Vidal, G. Laporte, and P. Matl, "A concise guide to existing and emerging vehicle routing problem variants," *European Journal of Operational Research*, vol. 286, no. 2, pp. 401–416, 2020.

[73] M. Štolba and A. Komenda, "The MADLA planner: Multi-agent planning by combination of distributed and local heuristic search," *Artificial Intelligence*, vol. 252, pp. 175–210, 2017.

[74] J. Tožička, J. Jakubův, and A. Komenda, "PSM-based Planners Description for CoDMAP 2015 Competition," in *Competition of Distributed and Multi-Agent Planners*, 2015, pp. 29–32.

[75] A. Torreno, E. Onaindia, and O. Sapena, "FMAP: Distributed cooperative multi-agent planning," *Applied Intelligence*, vol. 41, no. 2, pp. 606–626, 2014.

[76] S. S. Chouhan and R. Niyogi, "MAPJA: Multi-agent planning with joint actions," *Applied Intelligence*, vol. 47, no. 4, pp. 1044–1058, 2017.

[77] C. Boutilier and R. I. Brafman, "Partial-order planning with concurrent interacting actions," *Journal of Artificial Intelligence Research*, vol. 14, pp. 105–146, 2001.

[78] J. S. Penberthy and D. Weld, "UCPOP: A Sound, Complete, Partial Order Planner for ADL," *Proc. KR-92*, 1992.

[79] R. Brafman and U. Zoran, "Distributed Heuristic Forward Search for Multi-Agent Systems," in *ICAPS DMAP*, 2014.

[80] M. Crosby, M. Rovatsos, and R. P. Petrick, "Automated agent decomposition for classical planning," in *International Conference on Automated Planning and Scheduling*, 2013.

[81] S. Shekhar and R. I. Brafman, "Representing and planning with interacting actions and privacy," *Artificial Intelligence*, 2020.

[82] P. Schillinger, M. Bürger, and D. V. Dimarogonas, "Decomposition of Finite LTL Specifications for Efficient Multi-agent Planning," in *Distributed Autonomous Robotic Systems*, 2018, pp. 253–267.

[83] ——, "Simultaneous task allocation and planning for temporal logic goals in heterogeneous multi-robot systems," *International Journal of Robotics Research*, vol. 37, no. 7, pp. 818–838, 2018.

[84] Y. Chen, A. Stefanescu, and C. Belta, "Formal Approach to the Deployment of Distributed Robotic Teams Yushan," *IEEE Transactions on Robotics*, vol. 28, no. 1, pp. 158–171, 2012.

[85] A. Ulusoy, S. L. Smith, X. C. Ding, C. Belta, and D. Rus, "Optimality and robustness in multi-robot path planning with temporal logic constraints," *International Journal of Robotics Research*, vol. 32, no. 8, pp. 889–911, 2013.

[86] A. Nikou, D. Boskos, J. Tumova, and D. V. Dimarogonas, "Cooperative planning for coupled multi-agent systems under timed temporal specifications," *Proceedings of the American Control Conference*, pp. 1847–1852, 2017.

[87] F. Faruq, B. Lacerda, N. Hawes, and D. Parker, "Simultaneous Task Allocation and Planning Under Uncertainty," in *International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[88] R. Nissim and R. Brafman, "Distributed Heuristic Forward Search for Multi-agent Planning," *Journal of Artificial Intelligence Research*, vol. 51, pp. 293–332, 2014.

[89] H. Ma and S. Koenig, "Optimal target assignment and path finding for teams of agents," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, no. February, pp. 1144–1152, 2016.

[90] C. R. Garrett, T. Lozano-Pérez, and L. P. Kaelbling, "FFRob: Leveraging symbolic planning for efficient task and motion planning," *International Journal of Robotics Research*, vol. 37, no. 1, pp. 104–136, 2018.

[91] K. Hauser, V. Ng-Thow-Hing, and H. Gonzalez-Baños, "Multi-modal motion planning for a humanoid robot manipulation task," in *Robotics Research*, vol. 66, 2010, pp. 307–317.

[92] J. Barry, L. P. Kaelbling, and T. Lozano-Perez, "A hierarchical approach to manipulation with diverse actions," in *International Conference on Robotics and Automation*, 2013.

[93] W. Vega-Brown and N. Roy, "Asymptotically Optimal Planning under Piecewise-Analytic Constraints," *Algorithmic Foundations of Robotics*, vol. 12, pp. 528–543, 2016.

[94] M. Toussaint, "Logic-geometric programming: An optimization-based approach to combined task and motion planning," in *International Joint Conference on Artificial Intelligence*, 2015, pp. 1930–1936.

[95] D. Driess and M. Toussaint, "Hierarchical Task and Motion Planning using Logic-Geometric Programming," in *RSS Workshop on Robust Task and Motion Planning*, 2019.

[96] T. Lozano-Pérez and L. P. Kaelbling, "A constraint-based method for solving sequential manipulation planning problems," in *International Conference on Intelligent Robots and Systems*, 2014, pp. 3684–3691.

[97] C. Dornhege, P. Eyerich, T. Keller, S. Trüg, M. Brenner, and B. Nebel, "Semantic attachments for domain-independent planning systems," in *International Conference on Automated Planning and Scheduling Semantic*, 2009.

[98] E. Erdem, K. Haspalamutgil, C. Palaz, V. Patoglu, and T. Uras, "Combining high-level causal reasoning with low-level geometric reasoning and motion planning for robotic manipulation," in *International Conference on Robotics and Automation*, 2011, pp. 4575–4581.

[99] L. P. Kaelbling and T. Lozano-Pérez, "Hierarchical task and motion planning in the now," in *International Conference on Robotics and Automation*, 2011, pp. 1470–1477.

[100] S. Y. Lo, S. Zhang, and P. Stone, "PETLON: Planning efficiently for task-level-optimal navigation," *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems, AAMAS*, vol. 1, no. Aamas, pp. 220–228, 2018.

[101] F. Lagriffoul, D. Dimitrov, A. Saffiotti, and L. Karlsson, "Constraint propagation on interval bounds for dealing with geometric backtracking," in *IEEE International Conference on Intelligent Robots and Systems*, 2012, pp. 957–964.

[102] F. Lagriffoul, D. Dimitrov, J. Bidot, A. Saffiotti, and L. Karlsson, "Efficiently combining task and motion planning using geometric constraints," *International Journal of Robotics Research*, vol. 33, no. 14, pp. 1726–1747, 2014.

[103] S. Srivastava, E. Fang, L. Riano, R. Chitnis, S. Russell, and P. Abbeel, "Combined task and motion planning through an extensible planner-independent interface layer," in *International Conference on Robotics and Automation*, 2014, pp. 639–646.

[104] N. T. Dantam, Z. K. Kingston, S. Chaudhuri, and L. E. Kavraki, "Incremental task and motion planning: A constraint-based approach," in *Robotics: Science and Systems*, vol. 12, 2016.

[105] A. Akbari, Muhayyuddin, and J. Rosell, "Task planning using physics-based heuristics on manipulation actions," in *International Conference on Emerging Technologies and Factory Automation*, 2016.

[106] M. Toussaint and M. Lopes, "Multi-bound tree search for logic-geometric programming in cooperative manipulation domains," in *International Conference on Robotics and Automation*, 2017.

[107] I. Umay, B. Fidan, and W. Melek, "An integrated task and motion planning technique for multi-robot-systems," in *International Symposium on Robotic and Sensors Environments*, 2019, pp. 9–15.

[108] A. Akbari, F. Lagriffoul, and J. Rosell, "Combined heuristic task and motion planning for bi-manual robots," *Autonomous Robots*, vol. 43, no. 6, pp. 1575–1590, 2019.

[109] J. Motes, R. Sandstrom, H. Lee, S. Thomas, and N. M. Amato, "Multi-Robot Task and Motion Planning with Subtask Dependencies," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3338–3345, 2020.

[110] S. H. Semnani, H. Liu, M. Everett, A. De Ruiter, and J. P. How, "Multi-Agent Motion Planning for Dense and Dynamic Environments via Deep Reinforcement Learning," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 3221–3226, 2020.

[111] L. Cohen, T. Uras, T. K. Satish Kumar, and S. Koenig, "Optimal and bounded-suboptimal multi-agent motion planning," *Proceedings of the 12th International Symposium on Combinatorial Search, SoCS 2019*, no. SoCS, pp. 44–51, 2019.

[112] Z. Wang and M. Gombolay, "Learning to Dynamically Coordinate Multi-Robot Teams in Graph Attention Networks," *arXiv preprint*, 2019.

[113] E. Flushing, L. Gambardella, and G. Di Caro, "A mathematical programming approach to collaborative missions with heterogeneous teams," in *International Conference on Intelligent Robots and Systems*, 2014, pp. 396–403.

[114] H. Ravichandar, K. Shaw, and S. Chernova, "STRATA: unified framework for task assignments in large teams of heterogeneous agents," *Autonomous Agents and Multi-Agent Systems*, vol. 34, no. 2, pp. 1–25, 2020.

[115] A. Coles and A. Coles, "OPTIC," in *International Planning Competition 2018 Temporal Track*, 2018.

[116] A. Andreychuk, K. Yakovlev, D. Atzmon, and R. Stern, "Multi-agent pathfinding with continuous time," in *International Joint Conference on Artificial Intelligence*, 2019, pp. 39–45.

[117] A. Messing, G. Neville, S. Chernova, S. Hutchinson, and H. Ravichandar, "GRSTAPS: Graphically Recursive Simultaneous Task Allocation, Planning, and Scheduling," *International Journal of Robotics Research*, vol. 41, no. 2, pp. 232–256, 2022.

[118] J. Blazewicz, J. K. Lenstra, and A. H. Kan, "Scheduling subject to resource constraints: classification and complexity," *Discrete Applied Mathematics*, vol. 5, no. 1, pp. 11–24, 1983.

[119] P. Lamas and E. Demeulemeester, "A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations," *Journal of Scheduling*, vol. 19, no. 4, pp. 409–428, 2016.

[120] N. Fu, P. Varakantham, and H. C. Lau, "Robust partial order schedules for RCPSP/max with durational uncertainty," in *International Conference on Automated Planning and Scheduling*, 2016, pp. 124–130.

[121] P. Varakantham, N. Fu, and H. C. Lau, "A proactive sampling approach to project scheduling under uncertainty," in *AAAI Conference on Artificial Intelligence*, 2016, pp. 3195–3201.

[122] Y. Zhang and L. E. Parker, "Multi-robot task scheduling," in *International Conference on Robotics and Automation*, 2013.

[123] E. Bischoff, F. Meyer, J. Inga, and S. Hohmann, "Multi-Robot Task Allocation and Scheduling Considering Cooperative Tasks and Precedence Constraints," in *IEEE International Conference on Systems, Man and Cybernetics*, 2020.

[124] D. Matos, P. Costa, J. Lima, and A. Valente, "Multiple Mobile Robots Scheduling Based on Simulated Annealing Algorithm," in *International Conference on Optimization, Learning Algorithms and Applications*, 2021.

[125] P. Morris, N. Muscettola, and T. Vidal, "Dynamic control of plans with temporal uncertainty," in *International Joint Conference on Artificial Intelligence*, 2001, p. 499.

[126] I. Tsamardinos, "A Probabilistic Approach to Robust Execution of Temporal Plans with Uncertainty," in *Hellenic Conference on Artificial Intelligence*, 2002, pp. 97–108.

[127] M. Saint-Guillain, T. S. Vaquero, S. A. Chien, J. Agrawal, and J. Abrahams, "Probabilistic Temporal Networks with Ordinary Distributions: Theory, Robustness and Expected Utility," *Journal of Artificial Intelligence Research*, vol. 71, pp. 1091–1136, 2021.

[128] C. Fang, P. Yu, and B. C. Williams, "Chance-Constrained Probabilistic Simple Temporal Problems," in *AAAI Conference on Artificial Intelligence*, 2014.

[129] T. Vidal and H. Fargier, "Handling contingency in temporal constraint networks: From consistency to controllabilities," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 11, no. 1, pp. 23–45, Jan. 1999.

[130] J. Brooks, E. Reed, A. Graver, and J. C. Boerkoel, "Robustness in probabilistic temporal planning," in *AIII Conference on Artificial Intelligence*, 2015.

[131] P. Santana, T. Vaquero, C. Toledo, A. Wang, C. Fang, and B. Williams, "PARIS: A Polynomial-Time, Risk-Sensitive Scheduling Algorithm for Probabilistic Simple Temporal Networks with Uncertainty," in *International Conference on Automated Planning and Scheduling*, 2016.

[132] F. Habibi, F. Barzinpour, and S. J. Sadjadi, "Resource-constrained project scheduling problem: review of past and recent developments," *Journal of Project Management*, pp. 55–88, 2018.

[133] A. Kleywegt, A. Shapiro, and T. Homem-De-Mello, "The Sample Average Approximation Method for Stochastic Discrete Optimization," *SIAM Journal on Optimization*, vol. 12, no. 2, pp. 479–502, 2002.

[134] J. F. Benders, "Partitioning procedures for solving mixed-variables programming problems," *Numerische Mathematik*, vol. 4, pp. 238–252, 1962.

[135] W. Song, D. Kang, J. Zhang, and H. Xi, "Risk-aware proactive scheduling via conditional value-at-risk," in *AAAI Conference on Artificial Intelligence*, 2018.

[136] D. Duue and J. Pan, "An Overview of Value at Risk," *Journal of Derivatives*, vol. 4, no. 3, pp. 7–49, 1997.

[137] K. Stergiou and M. Koubarakis, "Backtracking algorithms for disjunctions of temporal constraints," *Artificial Intelligence*, vol. 120, no. 1, pp. 81–117, 2000.

[138] J. C. Beck and N. Wilson, "Proactive algorithms for job shop scheduling with probabilistic durations," *Journal of Artificial Intelligence Research*, vol. 28, pp. 183–232, 2007.

[139] N. Fu, H. C. Lau, P. Varakantham, and F. Xiao, "Robust local search for solving RCPSP/max with durational uncertainty," *Journal of Artificial Intelligence Research*, vol. 43, pp. 43–86, 2012.

[140] J. Luedtke, S. Ahmed, and G. L. Nemhauser, "An integer programming approach for linear programs with probabilistic constraints," *Mathematical Programming*, vol. 122, no. 2, pp. 247–272, Apr. 2010.

[141] J. Banfi, A. Messing, C. Kroninger, E. Stump, S. Hutchinson, and N. Roy, "Hierarchical Planning for Heterogeneous Multi-Robot Routing Problems via Learned Subteam Performance," *IEEE Robotics and Automation Letter*, 2022.

[142] D. Fried, S. E. Shimony, A. Benbassat, and C. Wenner, "Complexity of Canadian traveler problem variants," *Theoretical Computer Science*, vol. 487, pp. 1–16, 2013.

[143] P. Eyerich, T. Keller, and M. Helmert, "High-quality policies for the Canadian traveler's problem," in *AAAI Conference on Artificial Intelligence*, 2010.

[144] M. Barer, G. Sharon, R. Stern, and A. Felner, "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem," in *Symposium on Combinatorial Search*, 2014, pp. 19–27.

[145] M. C. Gombolay, R. J. Wilcox, and J. A. Shah, "Fast Scheduling of Robot Teams Performing Tasks with Temporospatial Constraints," *IEEE Transactions on Robotics*, vol. 34, no. 1, pp. 220–239, 2018.

[146] J. Blumenkamp, S. Morad, J. Gielis, Q. Li, and A. Prorok, "A Framework for Real-World Multi-Robot Systems Running Decentralized GNN-Based Policies," in *International Conference on Robotics and Automation*, 2022, pp. 8772–8778.

[147] Z. Wang, C. Liu, and M. Gombolay, "Heterogeneous graph attention networks for scalable multi-robot scheduling with temporospatial constraints," *Autonomous Robots*, vol. 46, no. 1, pp. 249–268, 2022.

[148] H. Raghavan, O. Madani, and R. Jones, "Active Learning with Feedback on Both Features and Instances," *Journal of Machine Learning Research*, vol. 7, pp. 1655–1686, 2006.

[149] Q. Li, W. Lin, Z. Liu, and A. Prorok, "Message-Aware Graph Attention Networks for Large-Scale Multi-Robot Path Planning," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 5533–5540, 2021.

[150] E. Seraj *et al.*, "Learning Efficient Diverse Communication for Cooperative Heterogeneous Teaming," in *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 1173–1182.

[151] W. Gosrich *et al.*, "Coverage Control in Multi-Robot Systems via Graph Neural Networks," *International Conference on Robotics and Automation*, pp. 8787–8793, 2022.

[152] X. Bresson and T. Laurent, "An Experimental Study Of Neural Networks For Variable Graphs," in *International Conference on Learning Representations Workshop Track*, 2018.

[153] G. Neville, S. Chernova, and H. Ravichandar, "D-ITAGS: A Dynamic Interleaved Approach to Resilient Task Allocation, Scheduling, and Motion Planning," in *ArXiv*, 2022.

[154] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Transactions on Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009.

[155] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A Comprehensive Survey on Graph Neural Networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 32, no. 1, pp. 4–24, 2021.

[156] V. P. Dwivedi, C. K. Joshi, T. Laurent, Y. Bengio, and X. Bresson, "Benchmarking Graph Neural Networks," *arXiv preprint*, 2020.

[157] C. K. Joshi, T. Laurent, and X. Bresson, "An Efficient Graph Convolutional Network Technique for the Travelling Salesman Problem," *ArXiv*, 2019.

[158] A. Cesta and A. Oddi, "Gaining efficiency and flexibility in the simple temporal problem," in *International Workshop on Temporal Representation and Reasoning*, 1996.

[159] L. Ford, "Network Flow Theory," RAND Corporation, Santa Monica, California, Tech. Rep. Paper P-923, 1956.

[160] M. Medress *et al.*, "Speech Understanding Systems* Report of A Steering Committee," *Artificial Intelligence*, vol. 9, no. 3, pp. 307–316, 1977.

[161] C. Skinner and S. Ranchurn, "The RoboCup rescue simulation platform," in *International Conference on Autonomous Agents and Multiagent Systems*, 2010, pp. 1647–1648.

[162] A. Paszke *et al.*, "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in *Advances in Neural Information Processing Systems 32*, Curran Associates, Inc., 2019, pp. 8024–8035.

[163] L. Gurobi Optimization, *Gurobi Optimizer Reference Manual*, 2022.

[164] D. P. Kingma and J. L. Ba, "Adam: A method for stochastic optimization," in *International Conference on Learning Representations*, 2015.

[165] Z. Wang and M. Gombolay, "Heterogeneous Graph Attention Networks for Scalable Multi-Robot Scheduling with Temporospatial Constraints," in *Robotics: Science and Systems*, 2020.

[166] W. Kool, H. Van Hoof, and M. Welling, "Attention, learn to solve routing problems!" In *International Conference on Learning Representations*, 2019.

[167] I. Solis, J. Motes, R. Sandstrom, and N. Amato, "Representation-Optimal Multi-Robot Motion Planning Using Conflict-Based Search," *IEEE Robotics and Automation Letters*, vol. 6, no. 3, pp. 4608–4615, 2021.

[168] S. Lavalle, *Planning Algorithms*. 2006.

[169] K. Hauser and V. Ng-Thow-Hing, "Fast smoothing of manipulator trajectories using optimal bounded-acceleration shortcuts," in *International Conference on Robotics and Automation*, 2010.

[170] T. Keller and P. Eyerich, "PROST: Probabilistic planning based on UCT," *International Conference on Automated Planning and Scheduling*, pp. 119–127, 2012.

[171] S. Garg, A. Bajpai, and Mausam, "Symbolic network: Generalized neural policies for relational MDPs," in *International Conference on Machine Learning*, 2020, pp. 3355–3365.

[172] F. Geißer, D. Speck, and T. Keller, "Trial-based heuristic tree search for MDPs with factored action spaces," in *International Symposium on Combinatorial Search*, 2020, pp. 38–47.

[173] Y. Carreno, J. H. A. Ng, Y. Petillot, and R. Petrick, "Planning, Execution, and Adaptation for Multi-Robot Systems using Probabilistic and Temporal Planning," in *International Joint Conference on Autonomous Agents and Multiagent Systems*, 2022, pp. 217–225.

[174] M. Rudolph, S. Chernova, and H. Ravichandar, "Desperate Times Call for Desperate Measures: Towards Risk-Adaptive Task Allocation," in *IEEE International Conference on Intelligent Robots and Systems*, Institute of Electrical and Electronics Engineers Inc., 2021, pp. 2592–2597, ISBN: 9781665417143.

[175] R. Van Der Krogt and M. De Weerdt, "Plan Repair as an Extension of Planning," in *Annual Conference on Artificial Intelligence*, 2005.